Conception d'architectures d'entrelaceurs parallèles pour les décodeurs de Turbo-Codes et de LDPC

Saeed Ur Reehman

THESE / UNIVERSITE DE BRETAGNE-SUD
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE-SUD
Mention :
Ecole doctorale SICMA

présentée par Saeed ur REHMAN
Préparée au Laboratoire Lab-STICC (UMR n°6285)
Université de Bretagne Sud Lorient, France

DESIGNING OPTIMIZED PARALLEL INTERLEAVER ARCHITECTURES FOR TURBO AND LDPC DECODERS

Thèse soutenue le 24 Septembre, 2014
devant le jury composé de :
Emmanuel Casseau (Rennes 1 / ENSSAT)
Fabienne Uzel-Nouvel (INSA Rennes)
Michel Jezequel (Telecom Bretagne)
Christophe Jego (ENSEIRB / IMS)
Philippe Coussy (Université de Bretagne Sud / Lab-STICC)
Cyrille Chavet (Université de Bretagne Sud / Lab-STICC)

Acknowledgment

All praises and thanks are for Allah, the Almighty who is the source of all knowledge and wisdom endowed to mankind and to the Holy Prophet Mohammad (Peace Be Upon Him) who showed light of knowledge to the humanity as a whole.

The achievement in this work involves high motivation and supervision of my Ph.D supervisors Philippe COUSSY and Cyrille CHAVET. I am thankful to them for guiding me to complete this thesis as it has been a great privilege working with the researcher of such a caliber. I have learnt a lot from their involvement in my professional as well as personal life. Their originality has triggered and nourished my intellectual maturity which will be beneficial for me forever. Now I can actually sense the beauty behind the research of new approaches. I express my gratefulness to prof. Emmanuel CASSEAU and prof. Fabienne NOUVEL for honoring us by accepting the difficult task of reviewing this thesis.

I would also like to thanks my parents, brothers and wife who motivated and supported me throughout. I would also like to acknowledge all my colleagues at Lab-STICC in France for making my PhD an enjoyable experience. In particular, I'd like to thank my colleague Dr. Awais Hussain SANI for his coordination during my research.

To my friends and family

Table of Contents

List of Figures

List of Tables

ABSTRACT

We live in the era of high data rate wireless applications (smart-phones, net-books, digital television, mobile broadband devices…) in which advanced technologies are included such as OFDM, MIMO and advanced error correction techniques to reliably transfer data at high rates on wireless networks. Turbo and LDPC codes are two families of codes that are extensively used in current communication standards due to their excellent error correction capabilities. For high throughput performance, decoders are implemented on parallel architectures in which more than one processing elements decode the received data. However, parallel architectures suffer from *memory conflict problem.* It increases latency of memory accesses due to the presence of conflict management mechanisms in communication network and unfortunately decreases system throughput while augmenting system cost.

To tackle memory conflict problem, three different types of approaches are used in literature. In first type of approaches, "architecture friendly" codes are constructed with good error correction capabilities in order to reduce hardware cost. However, these codes originate problem at the channel interleaver. In the second type of approaches, flexible and scalable interconnection network are introduced to handle memory conflicts at run time. However, flexible networks suffer from large silicon area and increased latency. The third type of approaches are design time memory mapping approaches in which the resultant architectures consist of ROM blocks used to store configuration bits. The use of ROM blocks may be sufficient to design parallel architecture that supports single codeword or single application. However, to design hardware architecture that supports complete standard or different applications, ROM based approaches result in huge hardware cost. To reduce hardware cost, optimizations are required to use as less ROMs as possible to support different applications.

In this thesis, we aim to design optimized parallel architectures. For this purpose, we have proposed two different categories of approaches. In the first category, we have proposed two optimized design time off-chip approaches that aim to limit the cost of final decoder architecture targeting the customization of the network and the use of in-place memory architecture.

In the second category, we have introduced a new method in which both runtime and design time approaches are merged to design flexible decoder architecture. For this purpose, we have embedded memory mapping algorithms on-chip in order to execute them at runtime to solve conflict problem. The on-chip implementation replaces the multiple ROM blocks

with a single RAM block to support multiple block lengths and/or to support multiple applications. Different experiments are performed by executing memory mapping approaches on several embedded processors.

Chapter 1

INTRODUCTION TO PARALLEL ARCHITECTURES FOR TURBO AND LDPC CODES

Table of Contents

In this chapter, error correction codes are discussed. Error correction codes can be classified into two broad categories: convolutional codes and block codes. Parallel hardware architecture is needed to support high throughput. Memory conflict problems which occur in parallel architectures implementation are introduced to highlight the importance of the work presented in this thesis.

1. Introduction

Wireless communication is undoubtedly one of the major research areas in telecommunication today. Broad progress can be observed in this field in the past decade, although it has been a topic of study since 1960s. The ongoing goal of providing enhanced services seamlessly and effectively continues to drive wireless communications. We have seen the outcomes in the form of cellular systems that have experienced exponential growth over the last decade with billions of customers worldwide. Incorporation of multimedia and value added services in telecommunication have dramatically increased data rate requirements.

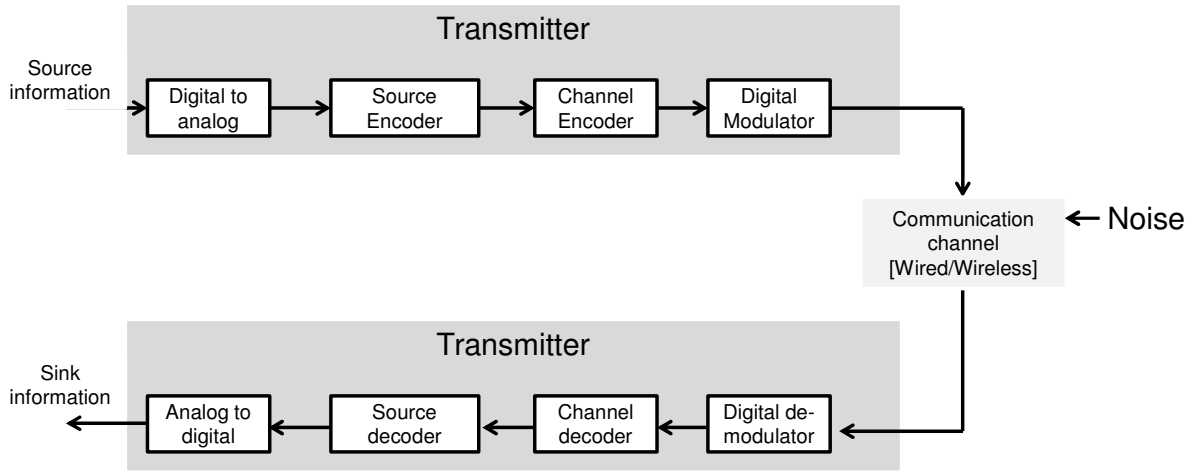


Figure 1. 1 A generic digital communication system

A generic digital communication system is shown in Figure 1. 1. The block diagram consists of building blocks each performing a certain well-defined operation on data. The transmitter transforms the signals from information source to the channel. The transmitter consists of blocks namely *Digital to analog converter, Source Encoder, Channel Encoder and Digital modulator*. *Digital to analog converter* converts the input information source into binary digits (bits), groups them together to form digital messages. *Source Encoder* tries to increase the information content of source symbols by removing the redundant information and encoding the source data using fewer bits than uncoded information. *Channel encoding* can reduce the errors rates at the expense of decoder complexity. Starting from K bits the channel encoder provides a codeword c of length N by adding N - K redundant bits. By means of introduced redundancy, the channel decoder is able to identify errors introduced by channel and correct some of them. *Digital Modulator* is a block that facilitates the transfer of information over a passband channel. In digital modulation, an analog carrier is modulated by a digital bit stream. Based on a particular application and channel condition, different

modulation techniques can be adopted, e.g. Phase Shift Keying (PSK), Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK) and Quadrature Amplitude Modulation (QAM)

The received channel information is transformed by the receiver to the sink information. The receiver consists of several blocks namely *Digital Demodulator, Channel Decoder, Source Decoder and Analog to digital converter*. *Digital Demodulator* takes the signal received from the Channel and converts it into digital data. *Channel Decoder* receives the decoded data from digital demodulator to detect errors introduced by Channel, corrects them and then it removes the redundant bits and extracts information words of *K* bits. *Source Decoder* performs the reverse operation of *Source Encoder* retrieving the same information as generated by information source along with *Analog to digital conveter*.

The channel encoder and decoder are responsible for the reliable transfer of data for which forward error correction (*FEC*) codes are widely used. *FEC* codes are among the significant parts of the whole system. *FEC* codes are described in the next section.

2. Forward Error Correction (FEC) Coding

We are now moving through the 4^{th} generation of wireless communication systems which are expected to achieve high data rates and reliable data transfer. Since error correction is one of the complex and power consuming part of whole transceiver design, therefore extensive research was carried out in the field of channel coding especially Forward Error Correction (FEC). Research in the field of *FEC* codes is aimed to find the best possible error correcting codes allowing high throughput decoding and their efficient VLSI implementation in term of area, speed and power consumption.

FEC codes can be classified into two categories: *Block codes* and *Convolutional codes*. *Convolutional codes* are increasingly used in different telecommunication standards due to their simple and efficiently implementable structures. Currently, convolutional codes are part of standards for mobile communication (HSPA [HSP04], LTE [LTE08]) and digital broadcasting (DVB-SH [DVBS08]). Convolutional codes work as a finite state machine which converts continuous stream of bits into continuous stream of coded bits.

In block codes, original information sequence is first divided into different blocks and then each block is independently encoded to generate code-word bits. The encoder must wait for the whole message block before starting the encoding step. However, in convolutional

encoder the code-word is transmitted as soon as encoding is started without any wait to obtain the entire message.

Two main error correcting codes families are used in current telecommunication standards. One from convolutional codes called Turbo codes and another from block codes called Low Density Parity Check (LDPC) codes. These two error correcting codes are widely used due to their excellent error correcting capabilities. However, implementation of decoders for these two codes for high data rate applications is a challenging task. In this thesis, we focus on the implementation of both of these codes on parallel architectures.

2.1 Introduction to Turbo codes

Due to their excellent error correction capabilities, Turbo codes [BER93] are part of most of the current telecommunication standards such as [LTE08] [HSP04] [DVBS08]. They are constructed through the parallel concatenation of two "simple" convolutional codes that share their information during the decoding step. The first convolution encoder encodes the message x in natural (original) order to produce $p^{(1)}$ parity bits, whereas the second one encodes the message in interleaved order (after passing the original message through interleaver) to generate $p^{(2)}$ parity bits. The output turbo codeword c is composed of the original message and parallel concatenation of parity bits.

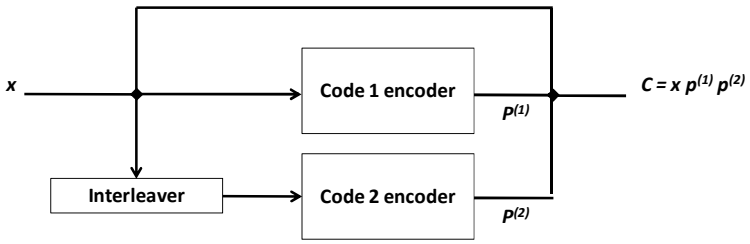


Figure 1. 2. Turbo Encoder

The high performances of turbo codes are due to the presence of this pseudo-random interleaver. Interleaving (Π) is a permutation law that scrambles data to break up neighbourhood-relations. It is a key factor for turbo-codes performances, which vary from one standard to another. The low-complexity iterative decoding algorithm for turbo-decoding makes its hardware implementation possible with the current standards. However, in order to achieve high throughput architectures, we will see that this interleaver generates memory access conflicts when parallel architectures are used.

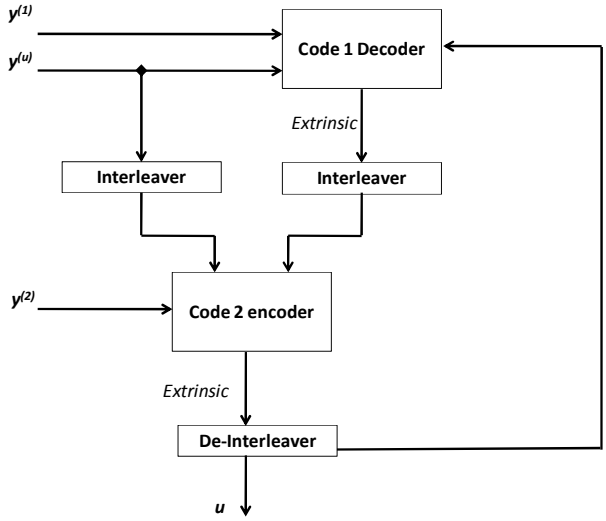


Figure 1. 3. Turbo Decoder

The turbo decoder receives input values $Y^{(u)}$, $Y^{(1)}$, $Y^{(2)}$ from the channel (resp. for x, $p^{(1)}$, $p^{(2)}$). One complete iteration of turbo decoder is carried out through two half iterations. Firstly *Code 1 Decoder* receives channel values for message bit $Y^{(u)}$, first parity bit $Y^{(1)}$ and deinterleaved extrinsic value from *Code 2 Decoder* to generate extrinsic value. Then, during the second half iteration, *Code 2 Decoder* creates extrinsic value from interleaved message bits, second parity bit $Y^{(2)}$ and interleaved extrinsic value from *Code 1 Decoder*. The final decision about the message bits is made based on the extrinsic values from the two decoders and channel values for message bits, after a fixed number of decoding iterations.

2.2 Introduction to LDPC codes

Low density parity check Codes (LDPC) are another class of very high performances error correction codes. They are members of the class of Block codes that are used to transmit information reliably through noisy communication channels. Many types of block codes are used in different applications such as Reed-Solomon [AHA], Golay codes [GOL61] or Hamming codes [HAM50]. LDPC codes have already been included in several wireless communication standards such as DVB-S2 and DVB-T2 [DVB08], WiFi (IEEE 802.11n) [WIF08] or WiMAX (IEEE 802.16e) [WIM06].

The code is represented with parity check equations. As a pedagogical example, consider a codeword: $\mathbf{C = [c_1\ c_2\ c_3\ c_4]}$ which satisfies the following three parity check equations.

$$c_2 \oplus c_3 \oplus c_4 = 0,$$
$$c_1 \oplus c_2 \oplus c_4 = 0,$$
$$c_1 \oplus c_3 \oplus c_4 = 0$$

In a LDPC code, codeword constraints (or parity check equations) are often expressed in matrix form as follows:

$$\underbrace{\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}}_{H} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

The above H matrix is an $M * N$ binary matrix where each row M_i of H corresponds to a parity check equation whereas each column N_j associated with codeword bit. A nonzero entry at $(i, j)^{th}$ location means that the j^{th} codeword bit is included in the i^{th} parity check equation. For a codeword $x \in C$ to be valid, it must satisfy all parity check equations.

LDPC codes can also be graphically represented as bipartite graph called *Tanner Graphs*. Such graphs depict the association between code (represented by variable nodes *VN*) bit and parity check equation (represented by check nodes *CN*). An edge e_{ij} connects the i^{th} check node with j^{th} variable node, if this variable node is checked by or included in i^{th} check node. This means that the edges of a Tanner graph are constructed with respect to the H matrix.

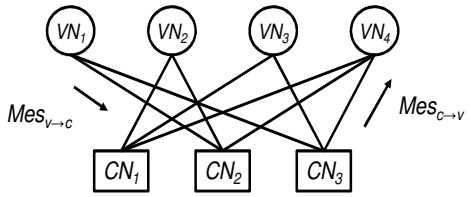


Figure 1. 4. Tanner Graph representation of H

Tanner graph is helpful in understanding the decoding process that exchanges messages between CN and VN ($Mes_{c\to v}$ or $Mes_{v\to c}$) along the edges of these graphs. These decoding algorithms are collectively called message-passing algorithms. They are a type of iterative decoding algorithm in which check nodes and variable nodes iteratively exchange messages until decoding is completed, such as belief-propagation or sum-product decoding [PEA88], min-sum decoding [FOS99] or normalized Min-Sum decoding [CHE02].

3. Memory conflict problem

In order to achieve high throughput performance, parallel hardware architectures are needed. The implementation of a typical parallel architecture is shown in Figure 1. 5. In this architecture, P processing elements (PEs) are used to process data elements which are connected to B memory banks through interconnection network, where $P = B$.

Unfortunately this kind of parallel architectures generates memory access conflicts as soon as several PEs simultaneously try to access to the same memory bank. This problem is also called "*collision problem*" [GIU02]. Memory conflict problem is a major source of concern in designing parallel architectures. The memory conflict problem is explained for turbo and LDPC codes in the two following sub-sections.

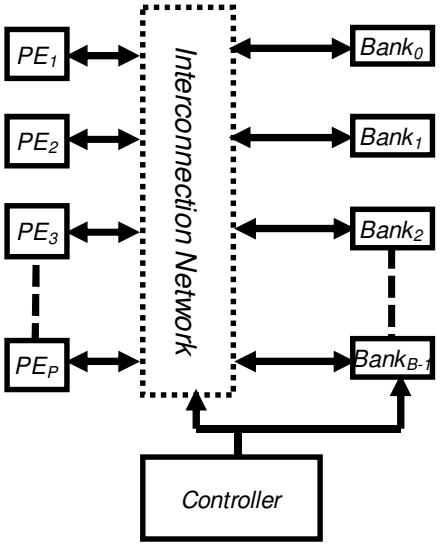


Figure 1. 5. Typical Parallel Architecture

3.1. Memory conflict problem for Turbo Codes

In parallel implementation of turbo codes, different number of processing elements access the data elements from the banks first in the natural order and then in interleaved order. The memory conflict problem for turbo codes is explained here with a pedagogical example. Let us consider $L = 20$, $P = B = 4$, $M = 5$ and $T = 10$, where L is the number of data elements, B is the number of memory banks, $M = L/B$ is the size of each memory bank and T is the total number of time accesses. The data elements are accessed first in natural order and then in interleaved order shown in Figure 1. 6.

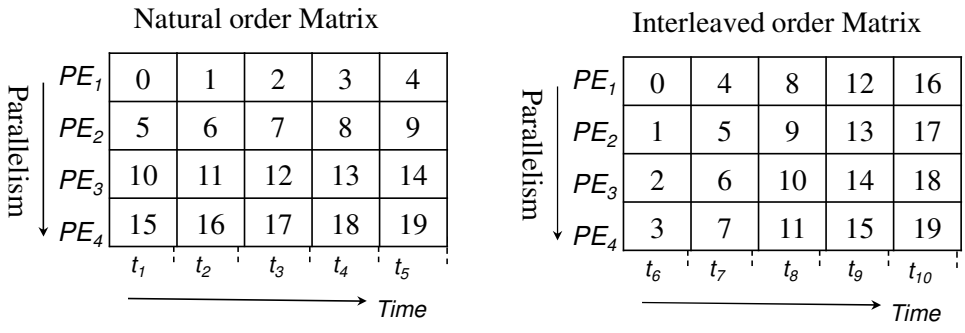
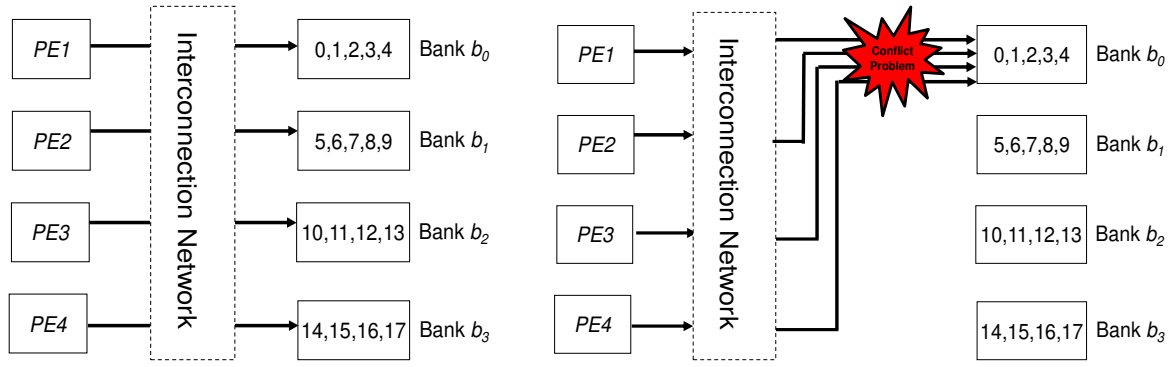


Figure 1. 6. Data access matrices for turbo codes

Let us consider that the data elements are stored in the banks in such a manner that all the processing elements always access different memory banks at each time instant in natural order as shown in Figure 1. 7.a e.g. data elements 0,5,10,15 are accessed by the processors in t_1 which are placed in different memory banks i-e b_0,b_1,b_2,b_3 respectively. Unfortunately, due to this memory mapping two or more processing elements need to access one particular memory bank at the same time instance in the interleaved order e.g. data elements 0,1,2,3 are accessed by the PEs in t_6 which are placed in the same memory bank b_0 as shown in Figure 1. 7.b. This issue is called *memory conflict problem* for turbo decoders.

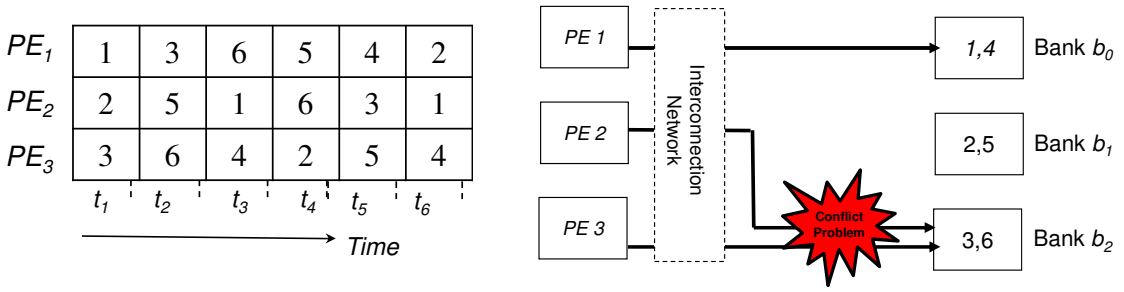


(a) Conflict free natural order Access (b) Conflict full Interleaved order Access

Figure 1. 7. Memory Conflict Problem in Parallel Turbo Decoder

3.2. Memory conflict problem for LDPC Codes

The memory conflict problem for the LDPC codes is different from turbo codes due to the difference in codes construction of these codes. The data access pattern of turbo codes are represented by natural and interleaved order matrices whereas LDPC codes are specified by their H matrices and represented by tanner graphs which shows that how data (*variable nodes*) must be processed by the processing elements (*check nodes*) in order to achieve good error correction performances. In order to explain a memory conflict problem in LDPC codes, let us consider $L = 6$, $P = B = 3$, $M = 2$ and $T = 6$ as shown in data access matrix in Figure 1. 8.a. The data elements stored in bank b_0, bank b_1 and bank b_2 are (1,4), (2,5) and (3,6) respectively. There is no conflict at time instances t_1 but at t_2 more than two processors want to access the same memory bank. Figure 1. 8.b shows the memory conflict for the time instance t_2.



(a) Data access matrix (b) Conflict at t_2

Figure 1. 8. Memory Conflict Problem in Partially Parallel LDPC Decoder

Several approaches exist in literature in order to tackle the memory conflict problem. However optimization is needed to design high throughput decoder architectures. The main purpose of this thesis is to optimize the design of parallel architectures focusing on the memory conflict problems (interleavers) in order to reduce the cost for high throughput Turbo and LDPC decoders. The rest of the thesis is organized as follow:

Chapter 2

In this chapter, an overview of the state of the art approaches to design parallel hardware architectures for Turbo and LDPC decoders is provided. The state of the art is presented in three different categories. The merits and limitations of each of the approaches are explored.

Chapter 3

In this chapter, we present our first approach that aims to limit the cost of final decoder architecture by targeting the customization of the network at the design time (off-chip). In the beginning, shuffled and non-shuffled turbo scheduling schemes are explored. Then, the proposed approach based on network relaxation method is described in details. Different experiments are performed for different test cases by using the proposed approach.

Chapter 4

In this chapter, our second contribution is presented. This approach is based on in-place memory mapping architectures in order to generate optimized hardware decoders at design time. We propose different algorithms based on Vizing theorem and transportation problem in order to solve memory conflict problem in polynomial time while providing optimized decoders.

Chapter 5

Finally, we present an on-chip approach to support multiple standards/applications in order to generate optimized hardware architecture. In order to avoid multiple ROM blocks needed to store controller information, we propose to embed memory mapping approaches on-chip such that complete multiple standards can be supported.

Chapter 2

STATE OF THE ART

Table of Contents

In this chapter, different state of the art techniques to tackle the memory conflict problem on parallel architectures for Turbo and LDPC codes are presented. The state of the art approaches are divided into three different categories: conflict free interleaving laws, run time conflict resolution and design time conflict resolution. Advantages and disadvantages of each technique are presented in order to motivate our work in this thesis. At the end, we explain the in-place and multiple read multiple write (MRMW) memory mapping architectures.

1. Introduction

Forward error correction codes are used for reliable data transfers between transmitters and receivers. In the associated decoders, parallel architectures are used to achieve high throughput performances. However, these kinds of architectures suffer from memory conflict problem. Many approaches are proposed in literature in order to overcome this issue. In this chapter, different approaches are discussed to implement parallel architectures taking into account the conflict problem for Turbo and LDPC decoders. These approaches can be classified in three categories.

In the first family of approaches, conflict free interleaving laws are defined. The goal is to construct codes with good error correction capabilities, reduced hardware cost and that allow avoiding memory conflicts. In the second family of approaches, conflicts are solved at run time by using flexible and scalable interconnection networks with sufficient path diversity (routing mechanism) and/or buffering techniques to handle memory conflicts. The third family of approaches deals with algorithms that assign data in memory in such a manner that all the processing elements can access memory banks concurrently without any conflict. These approaches which resolve the memory conflict problem at design time are referred as *memory mapping approaches*.

These families are described in detail in the three following sections.

2. Avoiding conflicts during the code construction

In the first category, memory conflict problem is taken into account during code construction. The main source of memory conflicts in Turbo codes comes from the interleaver. Hence, the aim is to develop conflict free interleaving laws with good error correction performance. Conflict free interleaving law provides parallel concurrent accesses to each memory bank without any conflict. An example of such solutions is proposed in [EMM03] in which spatial and temporal permutations are introduced to construct a conflict free interleaver. In order to explain this approach let us consider a block length of 16 data elements arranged row by row into a matrix (*initial matrix*) as shown in Figure2. 1.a. Interleaver leverages on two scrambling techniques: *temporal and spatial permutations*. The *temporal permutation* is obtained by changing the positions of the column in the initial matrix as shown in Figure2. 1.b. For *spatial permutation,* different circular permutations are performed to different columns to obtain the interleaved matrix as shown in Figure2. 1.c. The resultant matrix is a combination of temporal and spatial permutations. Each row is related to

a given processing element and the memory size is represented by the number of columns. The benefit of this approach is also the use of barrel shifter network which controller cost is very low. The Quadratic Permutation Polynomial (QPP) interleaver which is a part of current 3GPP LTE standard [LTE08] is mainly based on this idea.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
C_1	C_2	C_3	C_4

(a) *Initial matrix*

0	2	3	1
4	6	7	5
8	10	11	9
12	14	15	13
C_1	C_3	C_4	C_2

(b) *Temporal permutation*

	+ 4	+ 3	+0	+ 1
P1	4	10	3	13
P2	8	14	7	1
P3	12	2	11	5
P4	0	6	15	9
	C_1	C_3	C_4	C_2

(c) *Spatial permutation*

Figure2. 1. Interleaver construction

QPP interleaver was introduced in [YAN05] which is a *prunable and deterministic interleaver*. A *prunable interleaver* can be modified to obtain the interleaver of shorter length that keeps the error correction capabilities of the original larger interleaver. Prunable interleavers offer scalability in code-word to meet the channel conditions and changing user requirements. In deterministic interleaver, algorithms are used to produce on the fly addresses of interleaved data. The implementation of deterministic interleavers is easy as compared to *random interleavers* in which the addresses are generated randomly and for which dedicated memory (e.g. ROM) is required to store these addressing information. The performance QPP interleaver is near to random interleaver for long frame size whereas for short frame size, QPP interleaver outperforms random interleaver. QPP interleaver is represented by the following equation for a block size L:

$$\Pi(x) = (f_1 x^2 + f_2 x) \bmod L$$

where x and Π(x) represents the natural and interleaved address respectively and variables f_1, f_2 are different for different block lengths as specified in the standard.

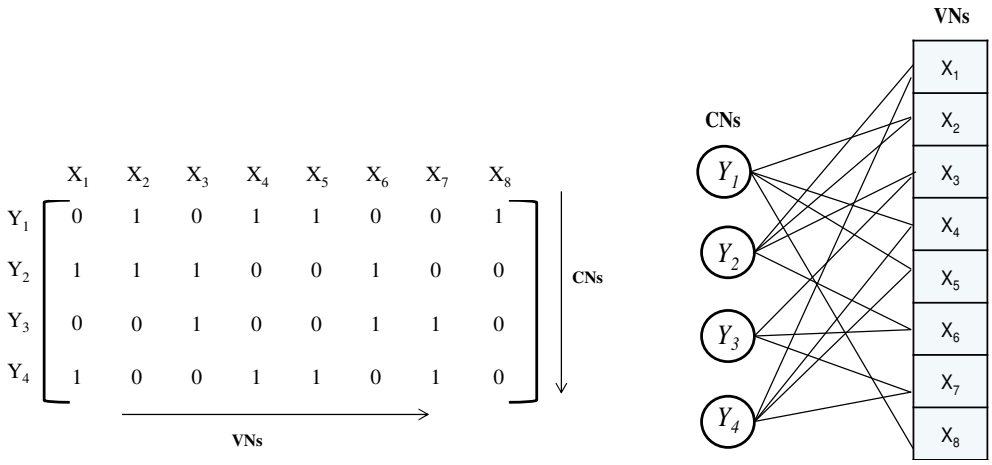
In [TAK06], QPP interleaver is proved to be contention-free for every window size that is a factor of interleaver length. However, it is not contention-free for other data rates in which high level parallelisms are used in the decoder.In [BOH07], a new interleaver for turbo codes is proposed. The interleaver is described by using four successive distinct laws and can also be defined using simple matrix ***S*** of size L x L where L is the size of the frame. This matrix is composed of k circularly right shifted z x z matrices (like in *LDPC* codes). The amount of shift is denoted by $\delta(r)$ and it is located at position P(r) of the matrix. In this approach, first the frame with $L = k.z$ elements is interleaved by z-row m-column permutations. Secondly, each group r (where r = [0, k-1]) of z elements is right shifted by $\delta(r)$ positions. Finally, in the last step the group of z elements is interleaved.

$$\left[\begin{array}{cccc|cccc|cccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

Figure 2. 2. Interleaver matrix

An example of such ***S*** matrix is shown in Figure 2.6 in which $L = 12$, $k = 3$, $z = 4$, P = [1,0,2] and δ = [1,3,0]. This interleaver has good performance and low-complex hardware implementation for high parallelized turbo decoders as compared to 3GPP-LTE interleaver. However, this interleaver is not a part of the current telecommunication standards.

The memory conflict problem for LDPC codes is handled by constructing *structured LDPC* codes [ZHA04] [MAN03]. *LDPC* codes are specified by their H matrices (see Figure 2. 3(a)) in which rows are associated to *check nodes* and columns to *variable nodes*. These codes can also be represented by *Tanner graphs* which show how data (*variable nodes*) must be processed by the processing elements (*check nodes*) in order to achieve good error correction performances (see Figure 2. 3(b)).



(a) H-matrix (b) Tanner graph

Figure 2. 3. Tanner graph formalization of an LDPC H-matrix

However, to achieve these excellent error correction capabilities proper construction of the *H* matrices is required. So *H* matrix must be constructed such that data transfer between check nodes and variable nodes can be made without any conflict for parallel architecture. In *structured codes,* the *H* matrix is divided into different blocks of sub-matrices (ZxZ matrices) where each sub-matrix is obtained by permuting rows of the identity matrix as shown in Figure 2.3. The check node processors access the vectors nodes data elements in parallel by using simple interconnection network like *barrel shifter* thanks to the structure of identity matrix in each sub-block.

Structured codes are part of current telecommunication standards such as IEEE 802.11n (WiFi) [WIF08] and IEEE 802.16e (WiMAX) [WIM06]. However, they only support one class of LDPC codes. A general approach to handle memory mapping problem is required to handle various existing and future classes of LDPC codes such as non-binary LDPC codes.

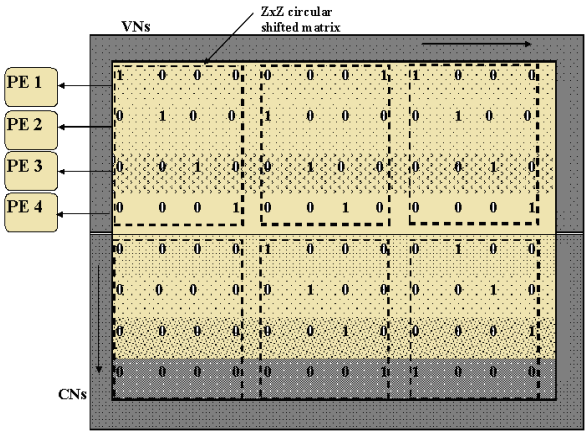


Figure 2. 4. Structured *LDPC* codes

Moreover, it must be observed that the conflict free data access order in the decoder part of the architecture can be different from the data access order coming from the channel as it is the case in QPP interleaver for example. This issue reports hence the conflict problem on the channel interleaver side.

3. Solving conflicts by means of dedicated runtime approaches

The second family of solutions solves memory access conflict problem by storing the data elements in different memory banks in an arbitrary order and then use additional mechanism (buffering/routing) in the interconnection network to manage conflicts at runtime. These approaches are referred as *run time conflict resolution. However, we have* referred these approaches as *time relaxation* methods in this document.

Such approach is presented in [WEH02] in which the data is simply stored in different memory banks without considering conflicting accesses and then additional buffers are used in the interconnection network to manage conflicts at runtime. This approach is based on a *LLR distributor* which is connected with all the *P* processing elements on one side and all the memory banks on the other side, as shown in Figure 2.4. The *LLR distributor* consists of interconnection network, buffers, *FIFOs* and multiplexers. The total cost and the latency of the architecture increases due to the use of buffers, FIFOs and multiplexers to manage conflict.

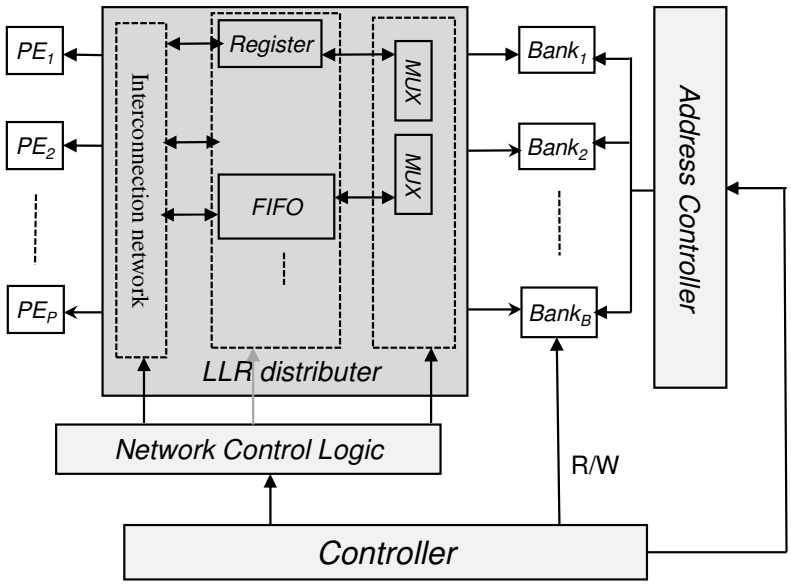


Figure 2. 5. Architecture based on LLR Distributor

In [WAN11], the authors proposed an approach based on Double-Buffer Contention-Free (DBCF) architecture. The *DBCF* architecture is built around the interleaver between the

processors and memory banks. This architecture consists of FIFOs associated with processors, circular buffers, multiplexers and bypass units as shown in Figure 2. 6. The conflicting accesses are routed into a dedicated circular buffer as soon as a conflict is detected. The interest of this approach has been demonstrated by designing an interleaver used in a HSPA+/LTE decoder. However, this architecture is configured on the basis of simulation results analysis in order to handle conflicts.

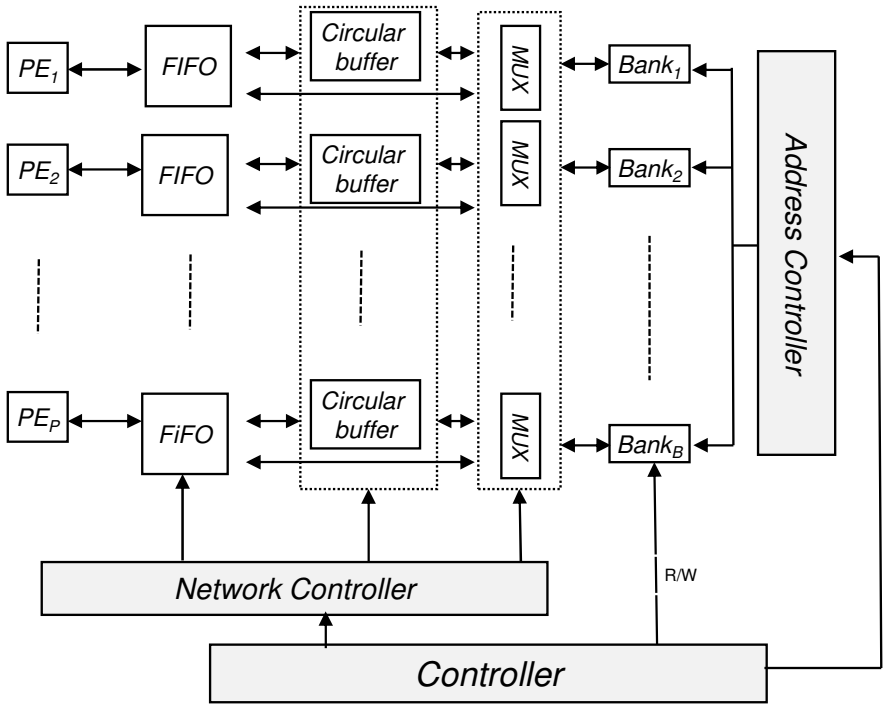


Figure 2. 6. Architecture based on Double buffer

Approaches based on *Network-on-Chip (NoC)* architecture are also proposed to resolve the conflicts on run time. In [NEE05], approaches based on *mesh*, *torus* and *cube* networks are proposed in which routers are used to contain packets for the destination information. However, these approaches suffer from reduced scalability to construct high throughput flexible on-chip communication network. Also, due to complex buffer management architecture to store conflicting data, the router complexity increases significantly with the increase of parallelism.

Another solution based on *NoC* oriented architecture is presented in [MOU07]. In this work (see Figure 2.7), the interconnection network can be configured on-the-fly to compete with any classical interconnection network such as *Butterfly* and *Benes*. *Butterfly network* has two main advantages: firstly, the network has huge scalability as a network of size *N* can be constructed from two networks of size *N/2*. Secondly, the packet routing algorithm is very

simple due to the use of destination address bits for selecting output port of router at each stage of the network. The routers stores conflicting packets using FIFOs. However, Butterfly network lacks in path diversity as it provides distinctive path between source and destination.

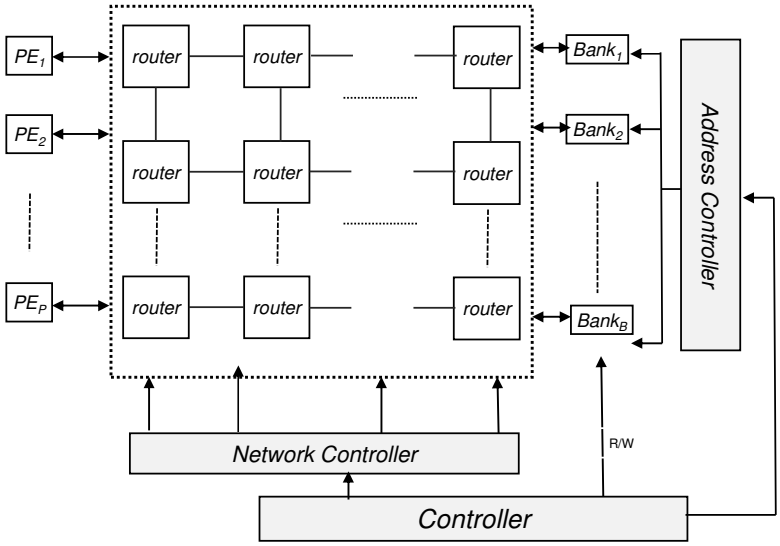


Figure 2. 7. Architecture based on NoC

Therefore, complex buffering architecture to manage conflicting packets is required which increases cost of the architecture. *Benes network* is the second multistage network studied in [MOU07]. It is constructed by concatenating two Butterfly networks back-to-back. The Benes network has good path diversity as it provides all possible permutations between inputs and outputs. However, this network avoids conflicts between packets, if all the packets have different destinations, which is not the case in turbo decoding. A modified topology and routing algorithm is proposed in this approach to optimize *Benes network* for turbo decoding. Routing algorithm transmits packets which are intended for different router at the same time and registers are used (instead of FIFOs) to store conflicting data. However, pre-processing is required to generate routing information and memory is needed to store router configuration.

A multistage network based on barrel shifter has been recently proposed for 3GPP LTE System for parallel decoder architecture in [WON10]. The connection between each memory bank and its corresponding processing element is established by shifting each sub block by a given offset due to the permutation characteristics of QPP interleaver used in LTE. The authors have presented a multistage interconnection network based on barrel shifter as shown in Figure 2.8. This figure shows the parallel architecture with multistage interconnection network for $P = 8$ with three stages. In the proposed network, 2^i bits are needed for shifting

data in the stage (3 − i) where $i = 0 \sim 2$ and the amount of shift in the stage (3 − i) is 2^i locations. So, we can compute that one bit is needed for stage 0, two bits are needed for stage 2 and four bits are needed for stage 3. Therefore, the three stages of the modified barrel shifter require driving seven bits for configuration at each access. Similarly, for $P = 4$, two stages of modified barrel shifter needs three bits.

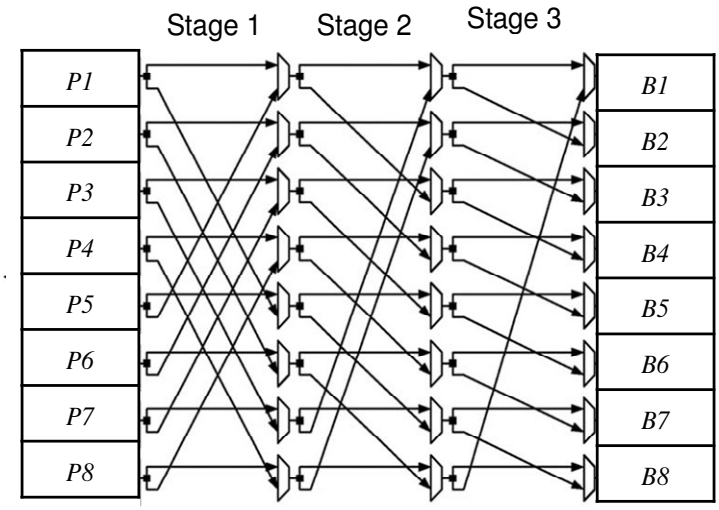


Figure 2. 8. Parallel architecture with multistage network

The data between the processor and memory is transferred immediately as the interconnection network has short path delay and simplified network control mechanism. However, the proposed approach can only be applied to QPP interleaver. This approach is not compliant with any other interleaving law.

The presence of interconnection network and buffer management mechanisms to manage conflicts increases hardware cost and latency of decoders which often restricts the implementation of such architectures for practical systems.

4. Solving conflicts with dedicated memory mapping approaches

The third family of solutions deal with memory access conflict problem by storing the data elements in different memory banks in such a way that all the processing elements can access to the data without any conflict at each time instance. Different mapping algorithms are proposed in state-of-the-art to perform pre-processing steps in order to determine each data element in the memories. These approach leverage on specific architectures that are presented in the last sub-section.

4.1. Memory mapping approaches

In [BEN04], one of the first algorithms based on simulated annealing meta-heuristic to resolve conflict problem for Turbo and LDPC codes is proposed. This algorithm is always able to find conflict free memory mapping, but the time to calculate the solution cannot be computed statically. Therefore, the computational complexity of the problem inhibits the addition of other constraints into the algorithm such as targeting a dedicated interconnection network.

In [JIN10], an approach based on optimized memory address remapping is presented. In this method certain collision-free exchange rules are defined to complete the simulated annealing procedure much faster than achieved in the traditional method presented in [BEN04] thanks to a reduced number of iterations to complete the annealing procedure . However, this method is also based on a meta-heuristic and the time of completion of the algorithm cannot be predicted.

In [CHA10a], a new simplified approach called Static Address Generation Easing (*SAGE)* is presented. This approach includes additional constraints to determine architecture oriented conflict free memory mapping. In SAGE, two *Mapping Matrices* (MAP_{Nat}, MAP_{Int}) are used during algorithm execution to store bank information. These matrices have the same order as the natural and interleaved order matrices as shown in Figure 2.9.

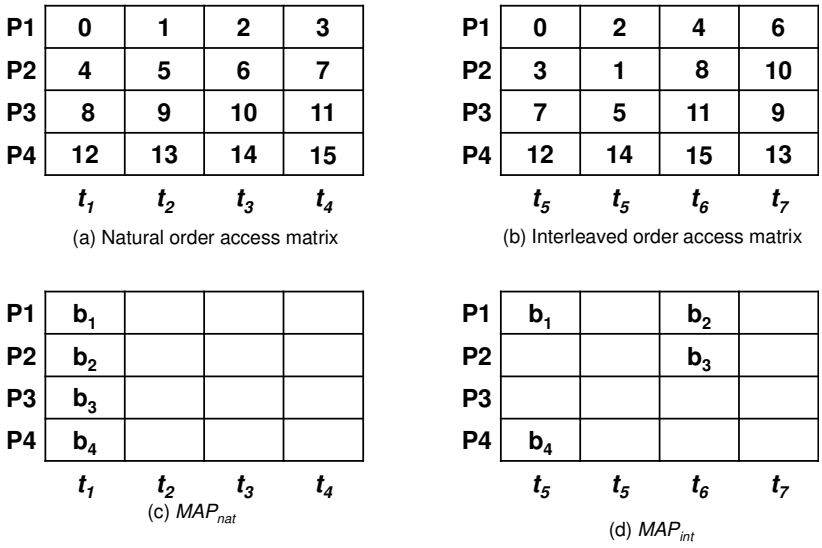
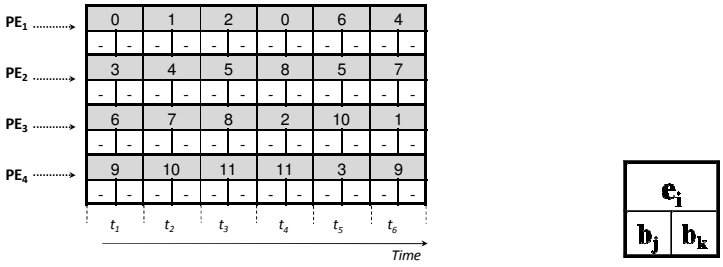


Figure 2. 9. Matrices used in *SAGE*

There are two constraints to be respected during the execution of the SAGE algorithm in order to find architecture oriented memory mapping. The first constraint is to allocate different memory banks to the cells of each column of the mapping matrices. The second

constraint is to respect the targeted interconnection network if supported by the interleaving law. The algorithm is initialized by assigning memory banks to the first column of M_{Nat} (see Figure 2.8.(c)). Then the entries corresponding to the data in M_{int} are updated (reported) with this mapping information (see Figure 2.8.(d)). In the next iteration, the most constrained column (i.e. the column with the greater number of assigned cells) is filled and reported in the same manner. This process continues until all the columns of the mapping matrices are filled with mapping information.

The above mentioned approach is limited to turbo-codes and a more generic approach has been proposed in [CHA10b]. This approach is able to solve memory conflict problem also for LDPC codes. It is based on Multiple Read Multiple Write (*MRMW*) mechanism in which each data element e_i consists of two memory locations: one for read data and the other for write data, as shown in Figure 2.9. For functional correctness, if data is accessed several times, then j^{th} read access of e_i must be equal to the $(j\text{-}1)^{th}$ write access of e_i. The algorithm assigns read and write memory banks to the most constrained column (i.e. the column with high number of data elements already mapped) of the *MAP* matrix. Then the corresponding entries in the other matrix are filled respecting the targeted interconnection network constraints. This process continues until *MAP* matrices are fully filled with targeted network constraints. However, recursion is needed when a conflict is detected. This approach innovates a new way to solve computationally complex problem through multiple read and multiple write mechanism. However, it can use backtracking making time to complete the algorithm unknown.



(a) Mapping matrix (b) An element of the matrix

Figure 2. 10. Multiple Read Multiple write (MRMW) approach

In [SAN11a], an approach based on Transportation problem modeling is presented. This method finds conflict free memory mapping for Turbo codes with architecture optimization. The mapping problem for turbo codes is transformed as transportation problem. The proposed approach is interesting as it is based on a polynomial time algorithm. However,

it works for a subset of cases and can be widely improved as demonstrated in the dedicated section of chapter-4.

In [SAN13], another polynomial time algorithm is presented to reduce the computational complexity to find conflict free mappings. The algorithm is based on two steps. In the first step, a bipartite graph is constructed based on two data access matrices. Then in the second step, a polynomial time bipartite edge coloring algorithm is used to find conflict free memory mapping. This approach can be used to solve memory conflict problem for both Turbo and LDPC codes.

Finally, in [BRI12] [BRI13a] [BRI13b], the authors have presented a memory mapping approach to find conflict-free memory mappings for both Turbo-codes and LDPC codes for any standard respecting targeted interconnection network. This approach is referred as *memory relaxation method* in this thesis. It is based on the idea of adding registers in the memory architecture (and not in the interconnection network) to deal with conflicting data and to respect the targeted interconnection network. However, the cost of the final architecture is increased due the inclusion of registers and their dedicated additional steering logic as shown in Figure 2.11. In this figure, the four processors are connected to four memory banks through a targeted interconnection network. Additional registers and steering logic are required to support the conflict free memory mapping with the targeted network as shown in Figure 2.11. However, if the targeted interleaving law is strongly incompatible with the targeted interconnection network the additional costs will be high.

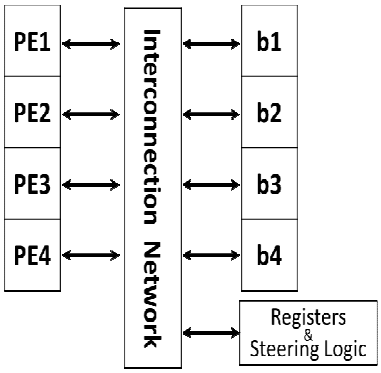


Figure 2.11. Resulting architecture with additional registers and steering logic for Memory relaxation based approach

4.2. Architecture for design time memory mapping approaches

All of the previously mentioned memory mapping approaches are either based on in-place memory mapping or multiple read multiple write (MRMW) memory mapping architecture as shown later in this section. We will describe the two kind of architectures in the two next subsections.

4.2.1. In-place memory mapping architecture

The architecture for turbo-like memory conflict problem is known as *in-place memory mapping architecture*. In order to explain in-place architecture, consider a set of L data elements $\{d_1, d_2,..., d_L\}$ and a set of P processing elements $\{PE_1, PE_2,..., PE_P\}$. These processors process each of the L data elements two times in T time instances $\{t_1, t_2,..., t_T\}$ first in natural order and then in interleaved order, where $T = 2L/P$. In order to store these L data elements and to achieve parallel processing of data for high throughput, a set of B memory banks $\{b_1, b_2,..., b_B\}$ are needed.

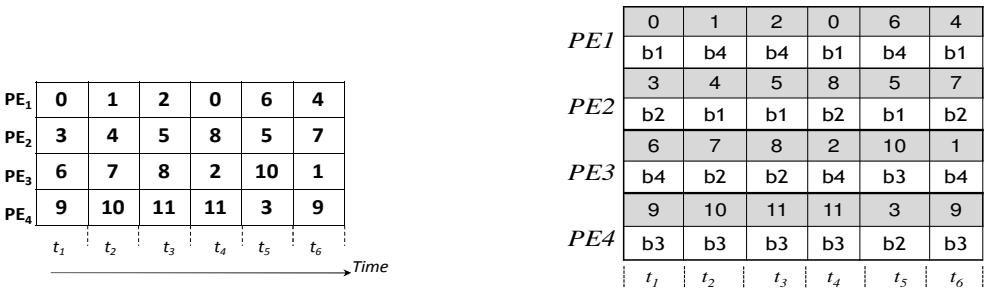
Mapping problem

We need to store the L data elements in B memory banks in such a way that P processing elements can access B memory banks in parallel for all time instances without any conflict.

For in-place conflict free memory accesses, the two following mapping constraints must be fulfilled:

- All memory banks have to be used only one time at each time instance (*conflict free*).
- Each data must be mapped in one and only one memory location (*in-place*).

As an example, a matrix in which each data is accessed twice (in natural and interleaved order), is shown in Figure 2.12.a. The resultant mapping of the considered example is shown in Figure 2.12.b by using memory mapping approach [CHA10a] which is based on in-place architecture. In this figure, each data in a column has a mapping cell which shows the memory bank from where a given data is read and written at a given time. It can be seen that every data element is read and written in the same memory bank, e.g. data 0 is read and written back in the same bank b_1.



(a)Example (b) Mapping matrix

Figure 2.12. In-place mapping

The architecture for resultant mapping can be seen in Figure 2.13 in which the data elements in each bank are shown. The processors read the data from the memory banks without any conflict by using memory controller through an interconnection network which is configured by network controller. The same configuration of a read cycle is also used for the write cycle.

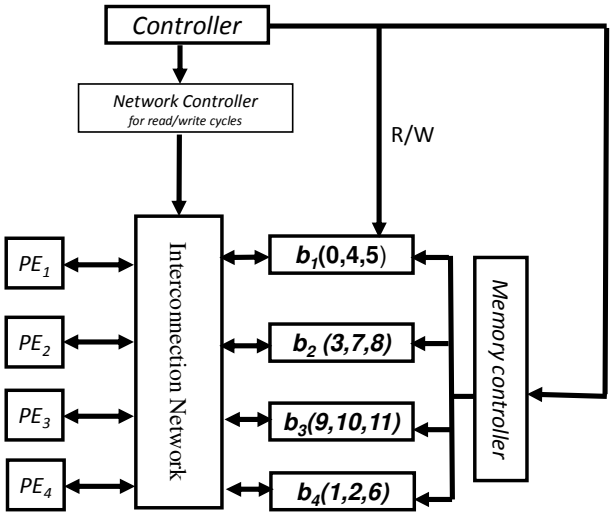


Figure 2.13. Resultant In-place mapping architecture

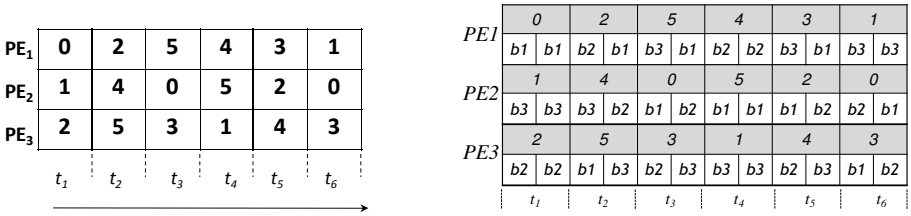
4.2.2. *MRMW* architecture

MRMW memory mapping architecture is used to solve LDPC-like memory conflict problems. The concept of *MRMW* memory mapping architecture is introduced in [CHA10b]. Then this concept is extended in [SANI13] using polynomial time algorithms to find conflict free memory mapping. In *MRMW* architecture, memory mapping of each data element is done in two memory locations: First location called *read mapping* represents read access to that data element whereas second location called *write mapping* expresses write access of that data element. The aim of *MRMW* architecture is to find memory mapping with optimal number of memory banks.

Mapping problem

Store L data in B memory banks in such a manner that P processing elements can access B memory banks at each time instance in parallel for first reading P data and then writing back these P data without any conflict.

The *MRMW* mapping is shown in Figure 2.14.b for the example in Figure 2.14.a by using [CHA10b]. In this figure data-2 at time instance t_2 is read from bank b_2 and written in a different bank b_1.



(a) Data access matrix (b) Resultant mapping

Figure 2.14. *MRMW* mapping

The resultant *MRMW* memory mapping architecture is shown in Figure 2.15. In this architecture the mapping of data elements in the banks is not mentioned because the mapping changes at each cycle according to the mapping shown in Figure 2.14.b. The network configurations for read operations are different from write operation. Therefore, configuration for read as well as write operations are needed which double the network controller cost. Hence, it can be noticed that the in-place mapping architecture could lead to reduce the network controller cost as compared to *MRMW* architecture.

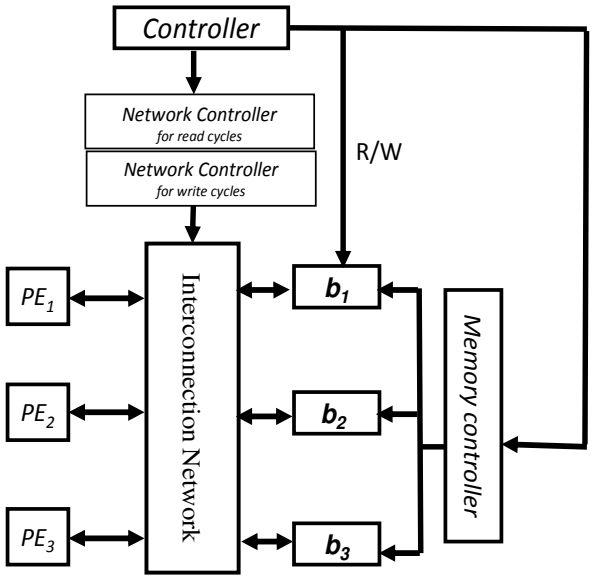


Figure 2.15. MRMW architecture

The *MRMW* architecture is more costly in term of area as compared to in-place memory architectures. Memory conflict problems like shuffled turbo (see chapter-3, section 2.2.2) and LDPC codes are solved using [CHA10b], [SAN11], [SAN13] based on *MRMW* architecture. So, in order to optimize generated architectures, it could be really interesting if in-place memory mapping architecture could be used to solve these problems.

5. Conclusion

In this chapter, different approaches to handle memory conflict problem for Turbo and LDPC codes are explained. One solution is to develop an interleaving law taking into account architectural constraints at the time of code construction. However, with this approach the memory mapping problem is only partially solved, and the designers still have to handle conflicts in the final architectures (for some parallelism degrees, block lengths, channel interleaver in telecommunication systems…).

A second technique exists in literature to tackle memory mapping problem at run time. However, the implementation of such architectures requires large hardware cost and latency due to the addition of conflict management mechanisms which limits its use in practical systems.

The third technique is to develop algorithms at design time that assign data in memory banks in such a manner that all processing elements can access their required data concurrently from memory without any conflict. Some of these techniques solve memory mapping problem for any type of interleaving law but results in costly hardware architecture. Others are limited to a subset of applications.

In this thesis, we aim to design optimized parallel hardware architectures to solve memory conflict problems. For this purpose, we propose two categories of complementary approaches. In first category, we propose memory mapping approaches that aim to limit the cost of final decoder architectures by customizing interconnection networks (see chapter 3) and by using in-place memory architectures (see chapter 4). In the second category, we propose to merge both runtime and design time approaches to design flexible decoder architectures. For this purpose, we have embedded polynomial time memory mapping algorithm on-chip along with the interconnection network in order to execute it at runtime to solve conflict problem (see chapter 5).

Chapter 3

OPTIMIZED MEMORY MAPPING APPROACH BASED ON NETWORK CUSTOMIZATION

Table of Contents

This chapter consists of two parts. First, a dedicated approach to explore the design space for parallel turbo decoder architectures is presented in which different configurations based on shuffled and non-shuffled schemes are considered. Then, thanks to the analysis of these experiments, we propose a new approach for conflict free memory mapping based on network customization to generate optimized architectures. This customization can be done by modifying a targeted network with additional network components if needed or by adding components starting from the scratch (directly connected wires). The proposed approach is compared with the state of the art approaches through different test-cases.

1. Introduction

In this chapter, we first present a dedicated approach to explore design space for hardware architectures of turbo decoders in order to analyze their hardware complexity. The turbo decoder memory issues are explored and state of the art approaches are used to solve the memory access conflicts in case of shuffled and non-shuffled turbo decoders. We have performed different experiments for a case study of turbo decoders for 3GPP-LTE.

The second part of this chapter is about our proposed memory mapping approach based on network customization. The complexity for a given memory mapping problem depends on the memory and network controller. Unfortunately, none of the existing approaches focused on network controller optimization to design conflict free memory mapping. Our proposal is to introduce a new approach based on finding conflict free memory mapping approaches which gives the degree of freedom in the interconnection network in order to reduce the complexity. In this approach, the interconnection network is customized to find a conflict free memory mapping which generates optimized architectures.

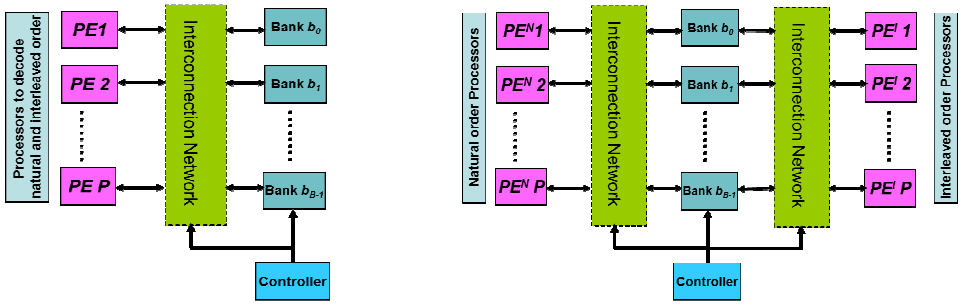
2. Dedicated approach to explore design space of turbo decoder architecture

Parallel turbo architecture can be based on different decoding techniques (shuffled, non-shuffled) and different scheduling like backward-forward, butterfly/butterfly-replica, … (see section 2.3). The impact of these techniques on the hardware complexity and throughput is usually determined at the end of design process after the synthesis process. Thus, the time to market is penalized and the probability of designing an optimized system decreases. In order to tackle this problem, we have introduced a dedicated approach to efficiently explore the design space of parallel turbo decoder architectures. Thanks to this approach, a tradeoff between the hardware complexity can be estimated for the architecture design process. The memory access conflict problem is solved using existing approaches in order to design high throughput architecture for any parallelism and interleaver. However, a penalty in terms of the hardware complexity is expected. This work has been carried out in collaboration with TELECOM Bretagne, Brest-France [SAC12].

2.1. Turbo decoder architecture

Figure 3. 1 shows the turbo decoder architecture. Through an interconnection network the processing elements (*PE*s) have access to a set of *B* memory blocks (single port RAM), allocated to keep the extrinsic information. The controller part consists of Read Only Memories which are used to address each memory block and control signals of the

interconnection network. The controller is also to be designed to address the ROM memories and to generate control signals of the memory blocks. Figure 3. 1(a) shows the architecture for non-shuffled decoding [BER90] in which all the *P* PEs are first assigned to decode the natural order, and then all of them are assigned to the interleaved order. However, as mentioned in [ZHA05], shuffled decoding can also be applied as shown in Figure 3. 1(b). In this case, *P/2* PEs are used to decode the natural order, while the remaining *P/2* PEs work on the interleaved order. Let *L* denotes the number of symbols in the received frame. This frame can be divided into *Q* sub-blocks that can be decoded in parallel. Each sub-block is formed by $M = L/Q$ symbols with $Q = P$ for non-shuffled turbo decoders and $Q = P/2$ for shuffled turbo decoders.



(a) Non-shuffled decoding Architecture (b) Shuffled decoding Architecture

Figure 3. 1 Decoding Architecture for Turbo Decoders

Let *T* denotes the number of clock cycles where each *PE* performs writing or reading memory access in order to execute one iteration (for a non-shuffled turbo decoder), or a half iteration (for a shuffled turbo decoder). Let *M* represents the size of each memory block. The size of each addressing ROM is $T*\lceil log_2(M) \rceil$. Note that iteration in a shuffled and non-shuffled turbo decoder takes the same time if both turbo decoders have the same sub-block size *M*.

2.2. Proposed design flow

The proposed design flow is detailed in Figure 3.2. Inputs include description of the interleaver law (π), the SISO decoder architecture (shuffled/non-shuffled), parallelism *P* and interconnection network delay (critical path of the interconnection network). The first step in the design flow is the generation of the data access description files based on the input information. These files contain the sequence of extrinsic information values that are read or written by each PE decoder at each clock cycle. This step also tackles memory issues in case of shuffled decoding (discussed later in details). The first step generates data access description file in which information about block length, parallelism and data access patterns are mentioned. In the second step, memory access description files are used to find conflict

free memory mapping using appropriate approaches. Thus, extrinsic values are assigned to *B* memory banks positions without any conflict. From this memory mapping, the controller can be designed and the content of ROM memories can be established. Finally, estimations of the turbo decoder throughput and hardware complexity are done.

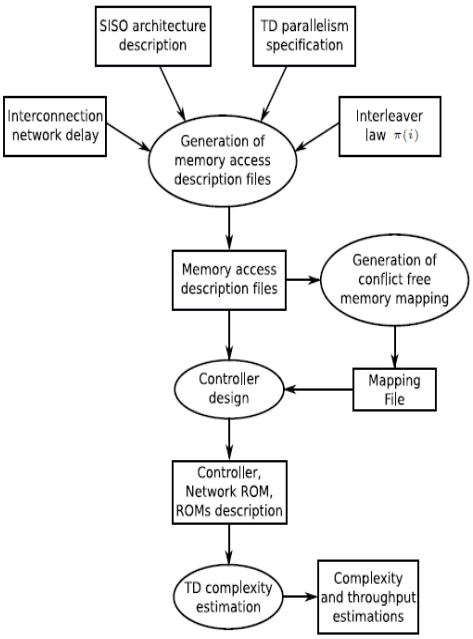


Figure 3. 2 Integrated design flow for Turbo decoder architectures exploration

2.2.1. Shuffled decoding memory issues

Shuffled turbo decoder architectures implementation presents additional memory constraints named *as concurrent access problem* and *consistency problem*. Here, we will give a brief description of these two memory issues and the way we have managed to tackle them.

Concurrent access problem occurs in shuffled turbo decoders when the extrinsic information for the same data is accessed in the natural as well as interleaved orders during the same clock cycle i.e. two processors access one data concurrently. In order to find the conflict free memory mapping, additional extrinsic memory locations are used to copy the extrinsic information with concurrent access ensuring that a correct exchange between natural and interleaved order is achieved. However, this kind of memory collision does not appear in non-shuffled architectures, since any information symbol is accessed first in natural order and then in the interleaved order.

The second memory issue for the shuffled decoding is *consistency problem*. *Consistency problem* occurs during the decoding process when a reading operation done by a *PE* in one domain (natural order) is followed by a reading operation performed by the *PE* in the other domain (interleaved order) in the next operation. This occurs for a writing operation as well.

As a result the data element can be overwritten as the two *PE* will write at the same memory location: if only one memory position is used for the extrinsic information, the same extrinsic value is read in both orders and the extrinsic value that has to be produced by one *PE* decoder is missed due to overwrite. In this case, performance degradation in the correction capabilities of the turbo decoders may occur. So, before finding conflict free memory mapping we need to add extrinsic memory locations as well due to consistency.

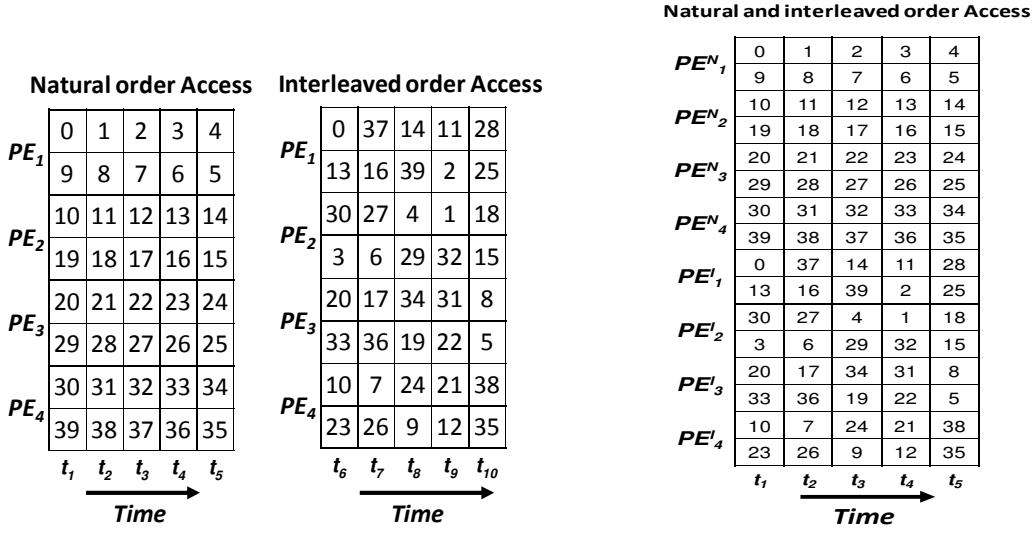
Let *l* denotes the number of additional memory positions used to solve concurrent access to the same extrinsic value and the consistency problems. Memory access description files then enable to carry out conflict free memory mapping which is presented later. Thus, L+*l* extrinsic values are assigned to *B* memory banks. From this memory mapping, the controller can be designed and the content of ROM memories can be established.

2.2.2. Solving memory conflicts

In-place memory mapping and multiple read multiple write (also called double) memory mapping are the two types of architectures that exist in literature. Non-Shuffled turbo decoding is similar to in-place memory mapping as data elements are first accessed in natural order and then in interleaved order. On the other hand shuffled turbo decoding presents double memory problem as the data elements are accessed in natural and interlaeaved order simultaneously. In order to find a conflict free memory mapping for these two problems, two approaches are introduced in this section: one is in-place memory mapping approach for non-shuffled decoding and another one is multiple read multiple write memory mapping approach for shuffled decoding in turbo decoders. Both approaches can be applied to find conflict free memory mapping.

Different in-place memory mapping approaches can be used to find conflict free memory mapping. In [SAN13], a polynomial time algorithm is presented in order to find conflict free mappings for conflict problem in turbo and LDPC codes. In-place architecture is used to solve problem for turbo like problems and *MRMW* architecture is used to solve memory conflict problem for LDPC like problems. Memory conflicts for non-shuffled turbo decoders is solved using [SAN13] using in-place architecture. Data access order can be illustrated through data access matrices as shown in Figure 3. 3 (a). In this figure, one matrix is related to the natural order access and the other one is related to the interleaved order access. In turbo decoders, Trellis level parallelism is used [WOO00] which are represented by radix-2^s (s = 1, 2, …) in which $d = 2^s$ rows are processed by each processors in parallel. Hence, each PE accesses *d* data elements i-e *d* rows in the matrix. Each matrix has *d.P* rows for the extrinsic values

accessed by the PEs, and *T* columns for the time instances. Data elements in each row are processed by the same PE. Similarly, the *d.P* data elements in each column have to be accessed in parallel by *P* PEs.



(a) Scheduling for non-shuffled decoding (b) Scheduling for shuffled decoding

Figure 3. 3 Scheduling for Turbo Decoding

As shown in Figure 3. 3 (b), the data access matrices for both natural and interleaved orders are processed concurrently for shuffled turbo decoders. Memory mapping with in-place architecture is not possible using [SAN13]. Therefore, the approach [SAN13] with *MRMW* architecture is used to solve memory mapping problem in this case.

The detailed description of the approach [SAN13] is given in section-4.1 of chapter-2 for in-place architecture and in section-4.2 of chapter-2 for *MRMW* architecture.

2.3. Case study: Turbo decoder for LTE

We have applied the approach described in the previous section to the 3GPP-LTE standard turbo decoder, with a frame size $L = 1024$ and a code rate $R = 1/2$. Parallelism is explored for $PE = 16$ *and* 32 that can use radix-2, radix-4 or radix-16. Shuffled and non-shuffled architectures are considered with schedule Butterfly-Replica and Butterfly [SAC12], respectively. In non-shuffled case there are $Q = 16, 32$ sub-blocks. In shuffled case, since the processors are distributed in both natural and interleaved orders, $Q = 8, 16$. Thus, we have considered sub-blocks sizes B = 32, 64 and 128. We have chosen six bits to represent the extrinsic information. The interconnection network is implemented by using B^* B Benes network [BNS65].

Table 3.1 Interconnection network area

Network size	Pipeline stages	Area (Logic gates)
32x32	1	7.4k
64x64	1	18.9k
128x128	2	45.7k

In the final architecture, the critical path is in the processor of the decoder. However, the critical path of the interconnection network is greater than the critical path of the decoder for some *B*. Therefore to prevent the critical path to be in the interconnection network, pipelining stages have been introduced. From logic synthesis results we determined that one pipeline stage is enough for $B = 32, 64$, while two stages are necessary for $B = 128$. Table 3.1 lists the number of logic gates required to implement the Beneš network for six bit width data after logic synthesis.

Table 3.2 Different configuration to explore the design space for turbo decoding

	Mode	Scheduling	Radix	Internal Memory
Config. 1	Non-Shuffled	Butterfly	2	YES
Config. 2	Non-Shuffled	Butterfly	4	YES
Config. 3	Non-Shuffled	Butterfly	16	YES
Config. 4	Non-Shuffled	Butterfly	2	No
Config. 5	Non-Shuffled	Butterfly	4	No
Config. 6	Shuffled	Replica	2	No
Config. 7	Shuffled	Replica	2	YES
Config. 8	Shuffled	Replica	4	No
Config. 9	Shuffled	Replica	4	YES

Nine turbo decoder configurations are selected and then studied for 16 and 32 PEs as shown in Table 3.2. The first 5 configurations were targeted for non-shuffled turbo decoders using different radix values. The last 4 configurations were defined to analyze the convenience of shuffled turbo decoders with radix-2 and radix-4. *PE* with and without internal memory were considered. Internal memory is a buffer that temporally stores extrinsic values which avoids a second access to the memory banks for a data element. Moreover, this buffer can alleviate collision problems since less memory accesses are necessary. The approach described in section 2.2.2 was applied to the 9 configurations shown in Table 3.2, for $P = 16$ *and* 32. The hardware cost of the resultant architecture was estimated using 90nm technology from STMicroelectronics in terms of NAND logic gate.

Figure 3.4 represents the equivalent number of logic gates for all the configurations with respect to the number of clock cycles necessary to decode a frame (directly related to the turbo decoder throughput). Configuration 1, 2 and 3, for 16 and 32 PEs, are Pareto-optimal

architectures. In non-shuffled turbo decoder configurations, the use of internal PE memories is convenient since it has a positive impact on the turbo decoder throughput (it reduce the number of reading accesses) and it helps to reduce the hardware complexity of the whole system. However, for the shuffled configurations, the turbo decoder hardware complexity is slightly reduced but the throughput is significantly affected.

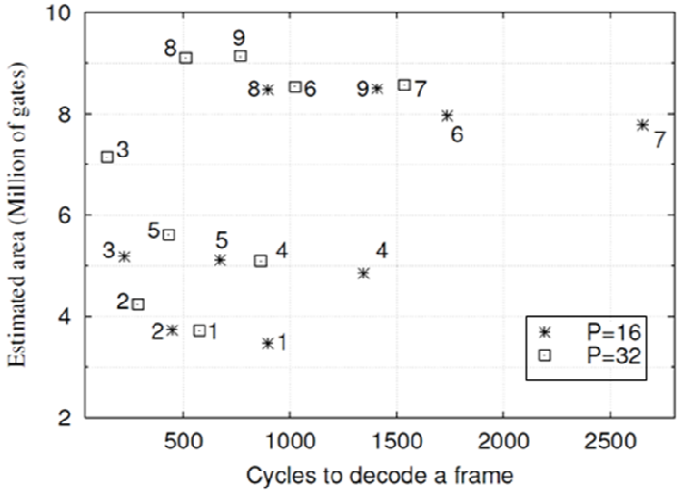


Figure 3. 4 Area estimations of the considered configurations

Table 3.3. We have not taken in to account the cost of processors as it is considered to be same in all the cases.). For configuration-5 the cost of network controller is 40% of the total cost and cost of memory controller is 50% of the total cost, whereas for configuration-8 the cost of network controller is 50% of the total cost and cost of memory controller is 37% of the total area. The cost of the interconnection network itself is just 1% of the total area. The analysis of the distribution of the estimated areas for in an in-place architecture is almost same for all configurations with in-place memory mapping (conf. 1 to 5), and the distribution of the cost of the different components in the *MRMW* architecture is almost same for all other configurations with *MRMW* memory mapping problem (conf. 6 to 9). Hence, it can be concluded that the network cost is negligible whereas network and memory controller is larger in the total cost.

Table 3.3 Cost calculation

	NW Cost	**NW Controller**	**Memory Controller**	**Extrinsic Memory**	**Total**
Config 5	18.9 k	2.1 M	2.5 M	0.4 M	5.1 M
Config 8	102.2 k	4.7 M	3.4 M	0.9 M	9.1 M

The approach used in [BRI13a] [BRI13b] is focused on memory mapping approach to design optimized parallel hardware architectures and to support a targeted interconnection

network. However, the cost of these approaches is very high in some cases when the interconnection network is not compatible with the interleaving law due to the additional components (buffers, register, multiplexers) needed to find conflict free access of data. We have proposed a memory mapping approach to design optimized parallel hardware architectures based on network customization which can reduce the network controller cost as network controller in the designed architecture is also among the most costly elements in the architecture. Moreover, the proposed approach is also able to generate a conflict free memory mapping with reduced cost for a targeted interconnection connection as compared to state of the art approaches.

3. Memory mapping approach based on network customization

The network and memory controller has a larger effect on the total cost as compared to other parts for the approaches used for finding conflict free memory mapping as shown in previous section. The *memory relaxation* approaches [BR12] [BRI13b] [BRI13a] emphasis on the optimization of memory and network controller. However, State of the art approaches exist which are based on customizing architectures to support the constraint of a particular targeted interconnection network. However, *Time relaxation* [WHE04] and *memory relaxation* [BRI13b] approaches are based on customizing architectures to support the constraint of a particular targeted interconnection network. In *memory relaxation* method, if the targeted interleaving law is strongly incompatible with the targeted interconnection network the additional costs are high. However in *time relaxation,* the final architecture is based on Benes network architecture and requires additional buffers which results in the increase of the total latency of the system. The targeted interconnection network has a large impact on the final cost of the architecture besides solving the parallel interleaving conflict as the set of possible permutations offered by the network can strongly restricts architectural design space exploration. Since this interconnection network has a great impact on the final architecture and it is considerably the cause of the problem which is not taken into account in the state of the art approaches. Since, the cost of the network controller depends on the size of the network (number of control bits is equal to the number of switches in the network) and it is not taken into account in the state of the art approaches. A smart memory mapping approach should focus on this network directly in order to adapt the network constraint to the interleaving law as much as possible. In this way, the optimization of the generated architecture will be more impressive than the existing approaches as it will be seen with proposed approach: we call it *network relaxation*. This work has been published in [REH14b].

3.1. Proposed Approach

3.1.1. Memory Mapping with Network Relaxation

Our proposed approach aims to take advantage of network relaxation principle. Figure 3. 5 presents an approach by considering the customization of the interconnection network and reducing the cost of the controller architecture. The constraint relaxation is provided by modifying the original network by adding additional multiplexers/switches. The idea is to keep the advantage of memory mapping approaches like [CHA10a] or [TAR04] in terms of architectural cost and latency, while proposing an approach that is able to target any application as [WHE04] or [BRI12].

First, starting from the description of an interleaving law (number of data, interleaving algorithm, parallelism) and a targeted interconnection network (i.e. NULL, Barrel-Shifter(BS), Butterfly(BF), Benes(BEN), Cross-Bar(CB)), the set of input memory mapping constraints is generated and provided to our memory mapping algorithm. Then, the next step consists in applying a memory mapping algorithm under network constraint. In the network library, all the permutations offered are stored separately for each network. This mapping step aims to fully explore the memory mapping solution space by checking all the permutations of the selected network. If no memory mapping solution exists for this network, then the set of permutations will be extended by addition of a network component, resulting into customized network architecture with enriched set of permutations (see Figure 3. 8). At the end the resulting architecture is generated. By applying this process for all available networks in the library, the designer is able to widely explore the design space and to select the best solution.

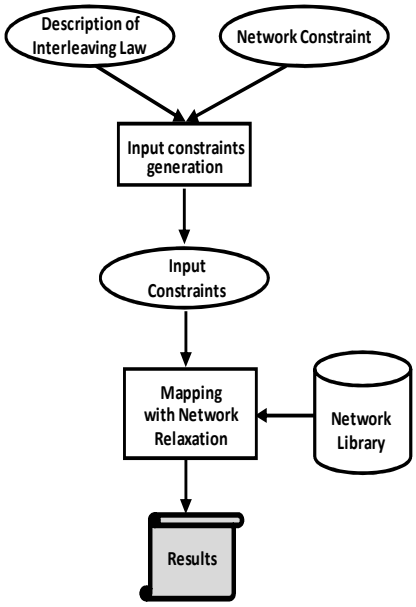
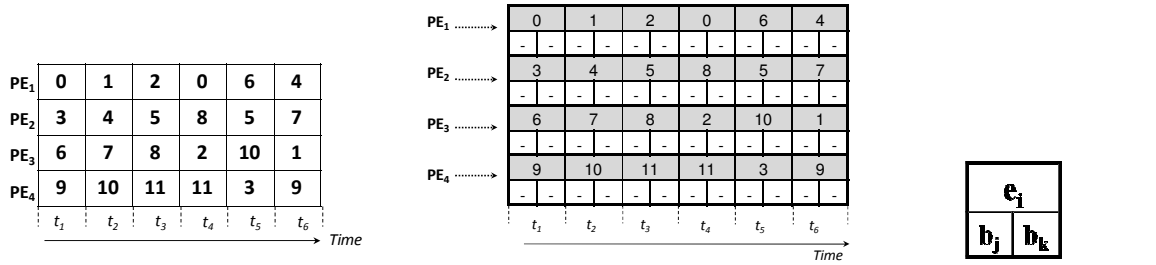


Figure 3. 5. Proposed Memory Mapping Exploration Flow for Network Relaxation Approach

The proposed algorithm is based on the memory mapping model proposed in [CHA10b]. A formal description of the mapping matrix model is presented here. Let us consider a parallel decoder architecture composed of P processing elements $PE=\{PE_1,\ldots PE_p\}$ and $P= B$ memory banks $B=\{b_0,\ldots b_{B-1}\}$ to store L data. We need $P= B$, because it is always possible to find conflict free memory mapping using $P= B$ for any conflict problem [CHA10a][CHA10b] and the increase in the number of banks will result in increase of the size of network which will increase the cost of the network controller. Figure 3.6(a) represents a data access matrix for a parallelism $P=4$ which is scheduled as a table. Lines match processing elements, i.e. each line represents all the data processed by its associated PE. Columns represent computation instants needed to process $L=12$ data elements.



(a) Data access matrix (b) Mapping matrix (c) Element of matrix

Figure 3. 6. Memory Mapping model

The data access matrix in then modeled to a mapping matrix as shown in Figure 3. 6(b) in which each data is associated to two mapping cells which will be filled with the label of a memory bank. An element of the memory mapping matrix is shown in Figure 3. 6(c) in which data e_i is read in the memory bank b_j, and written in the memory bank b_k after having been processed by a PE. In order to guarantee a valid memory mapping, constraints have to be respected for a given parallelism and interleaving law.

Memory constraints:

1- Data processed at the same cycle (i.e. data that are read or written concurrently at time instance) have to be stored in different memory banks.

2- The i^{th} read access to a given data must be performed in the same memory bank such that its $(i-1)^{th}$ write access i.e. a data must be read in the same memory location it has been written.

Network objective:

The memory mapping has to respect the set of supported permutations (i.e. this set is initialized with permutations of user-defined network constraint topology).

```
Start ;    // Boolean value initialized with TRUE
S_Perm ;   // Set of possible permutations in the selected network
M_Map;     // Memory Mapping matrix
C_i;       // Selected column i in M_MAP
LVP_Ci;    // List of valid permutations for column C_i
VP_Ci ;    // An element of LVP_Ci
Algorithm Map&NetRelax (S_Perm, M_Map, Start)
C_i = SelectColumn(M_Map);
LVP_Ci = SelectValidPerm(S_Perm, C_i, M_Map);
If ((LVP_Ci is not empty) and not (FullyMapped(M_MAP))) Then
   Do
     VP_Ci = Select&RemoveFirstPerm(LVP_Ci);
     MapColumn(C_i, VP_Ci);
     Start = FALSE;
     Map&NetRelax (S_Perm, M_Map, Start);
     If ((Start = FALSE) and not(FullyMapped(M_MAP))) Then
        RemoveMapColumn(C_i, VP_Ci);
     End if;
   While ((Start = FALSE) and (LVP_Ci is not empty) and
         not(FullyMapped(M_MAP)));
  If ((Start = FALSE) and not(FullyMapped(M_MAP)))  Then
      AddNewNetComp(S_Perm);
      EraseMap(M_Map);
      Start = TRUE;
      Map&NetRelax (S_Perm, M_Map, Start);
  End if;
End if;
```

Figure 3. 7. Mapping algorithm with network relaxation

The proposed memory mapping algorithm (cf. Figure 3. 7), first selects the most constrained column (i.e. the column with high number of data elements already mapped) in the memory mapping matrix (all the matrices are initially empty, so the mapping algorithm starts from the first column). Then the algorithm generates the subset of valid permutations for this column from the set of supported permutations in the targeted interconnection network. If this subset is not empty (i.e. conflict free memory mapping solution could be obtained for the selected column, with respect to the set of possible permutations) then each possible permutation from this subset is explored one by one, until a final conflict-free memory mapping is generated by our recursive algorithm. If no such valid memory mapping can be found with the targeted network then the set of possible permutations must be extended

by adding a new network component (i.e. multiplexer or switch) to the existing network. In that case the memory mapping algorithm restarts from the beginning with this new extended set of permutations. The algorithm keeps on customizing the network by adding switch to the network in each iteration until a fully connected Benes network is constructed (worst case).

As shown in Figure 3. 8(a), the algorithm starts by finding conflict free memory mapping with one switch and if the mapping is not possible another switch is added to the network (see Figure 3. 8(b)). Afterwards, the algorithm adds another switch to the network if the mapping is not possible with previous network and tries to find conflict free memory mapping using permutation resulted by this network (see Figure 3. 8). This process continuous (see Figure 3. 8(d) to (f)) until a fully connected network is established which can always give a conflict free memory mapping according to [TAR04]. However, during this process from Figure 3. 8(a) to (e), if the algorithm succeeded to find conflict free memory mapping then the network at that stage will be the optimized resultant network.

The architecture generated with network relaxation is composed of a classical interconnection network (i.e. Barrel-Shifter, Butterfly…) or NULL network along with additional network component(s) named customized network. Then, since the source of the memory conflicts is relaxed, it is possible to find an optimized conflict free architecture compared to [WHE04] or [BRI12].

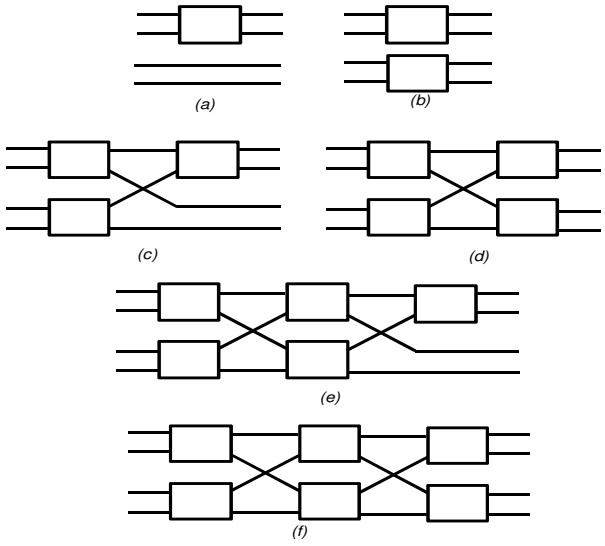
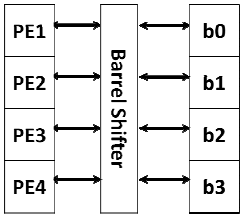


Figure 3. 8. Network customization

3.1.2. Pedagogical Example

In order to fully elaborate the proposed approach, the data access pattern example shown in Figure 3. 6(a) is considered. Firstly, a Barrel-Shifter BS is considered as input network constraint for this example. Hence, the set of permutations offered by a BS (see Figure 3.

9(b)) is selected from the library. The general architecture targeting BS is shown in Figure 3. 9(a).



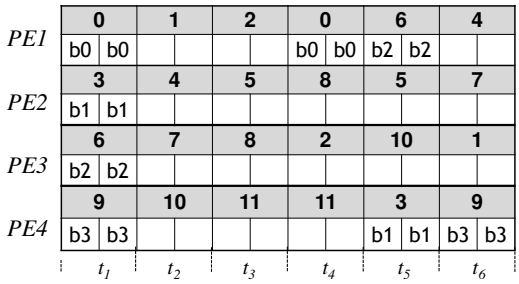
(a) Architecture

σ1	σ2	σ3	σ4
b0	b3	b2	b1
b1	b0	b3	b2
b2	b1	b0	b3
b3	b2	b1	b0

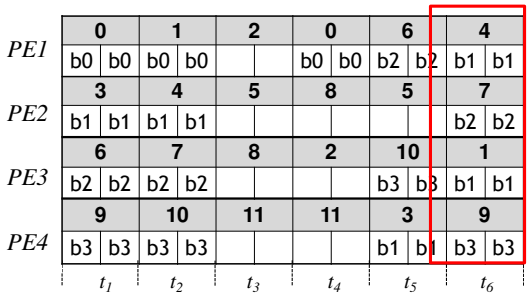
(b) Permutation

Figure 3. 9 Barrel shifter

The mapping is started from the 1st column and the column is filled according to the permutation supported by the BS (σ1 of Figure 3. 9(b)). This column is then reported in the mapping matrix accordingly as shown in Figure 3.10(a).



(a) Initial



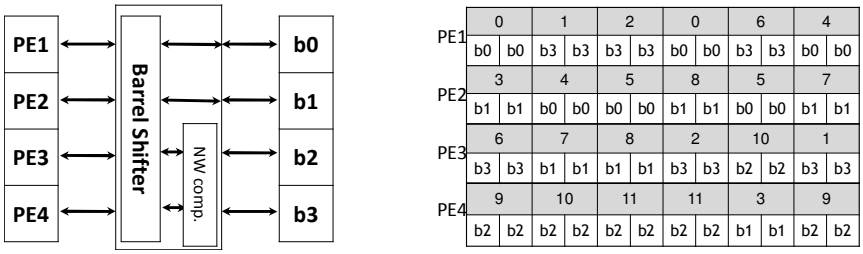
(b) Problem while mapping with BS

Figure 3. 10 Mapping with BS

Table 3.4 Permutations after adding network component with BS

σ1	σ2	σ3	σ4	σ5	σ6	σ7	σ8
b0	b3	b2	b1	b0	b3	b2	b1
b1	b0	b3	b2	b1	b0	b3	b2
b2	b1	b0	b3	b3	b2	b1	b0
b3	b2	b1	b0	b2	b1	b0	b3

It can be seen that after one iterations of our algorithm, the partial memory mapping described in Figure 3.10(b) is achieved. At this step, when the second column is filled according to the permutations of BS and when this column is reported, the permutation at the last column is not supported by BS permutations as shown in Figure 3.10(b) as the last column has two data elements (4 and 1) that are assigned to one banks b_1 (i.e. this results in an invalid mapping).So an additional network component (i.e. a switch which allows swapping input data) is needed with the Barrel-Shifter network and an extended set of permutations is generated as shown in Table 3.4. The position of the switch in the network will depend on the structure of the *Benes network* (because it is always possible to find conflict free memory mapping with a fully connected *Benes network* [TAR04][CHA10b]).

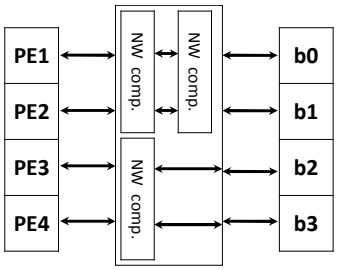


PE1	0		1		2		0		6		4	
	b0	b0	b3	b3	b3	b3	b0	b0	b3	b3	b0	b0
PE2	3		4		5		8		5		7	
	b1	b1	b0	b0	b0	b0	b1	b1	b0	b0	b1	b1
PE3	6		7		8		2		10		1	
	b3	b3	b1	b1	b1	b1	b3	b3	b2	b2	b3	b3
PE4	9		10		11		11		3		9	
	b2	b2	b2	b2	b2	b2	b2	b2	b1	b1	b2	b2

(a)Architecture (b) Mapping

Figure 3. 11 Network relaxation with BS

Then, the memory mapping process is done with this new set of permutations. The entire matrix can be filled according to this new set of permutations as shown in Figure 3. 11(b). The resultant architecture based on customized network is shown in Figure 3. 11(a) which consists of BS network and one additional network component (a swicth).



PE1	0		1		2		0		6		4	
	b0	b0	b2	b2	b2	b2	b0	b0	b3	b3	b0	b0
PE2	3		4		5		8		5		7	
	b2	b2	b0	b0	b0	b0	b3	b3	b0	b0	b3	b3
PE3	6		7		8		2		10		1	
	b3	b3	b3	b3	b3	b3	b2	b2	b1	b1	b2	b2
PE4	9		10		11		11		3		9	
	b1	b1	b1	b1	b1	b1	b1	b1	b2	b2	b1	b1
	t_1		t_2		t_3		t_4		t_5		t_6	

(a)Architecture (b) Mapping

Figure 3. 12 Network relaxation without any NW constraint

If no network constraint is defined by the designer, the algorithm starts from scratch considering no network constraints, i.e. start mapping with NULL network (each PE is directly connected to a single memory bank through a wire). The algorithm will add new network components to the network when needed, until a valid memory mapping is achieved. As a result a fully customized network is developed. The complete mapping for the considered example using network relaxation can be seen in Figure 3. 12. The algorithm will find the conflict free memory mapping for all the possible combination of the available permutations (i-e 1st column is filled with σ1 and if the mapping is not valid, then σ2 is tried and so on).

3.2. Experiments and Results

Currently turbo codes are used in different standards. However, interleavers used in these standards are not conflict free for every type of parallelism. The proposed approach is able to find conflict free memory mapping for any type of interleaver and for any type of parallelism. This section presents the different experiments we performed to show the interest of the proposed approach. All the results in this thesis are given in NAND-gate equivalent

area using 90nm technology from STMicroelectronics. These estimations are based on synthesized and pre-characterized components (Registers, multiplexers, …). The number of the different components is provided by the mapping tool and as a result the estimations for the architecture are generated. We performed experiments for two test cases: HSPA and LTE. In all these results we have considered the network controller cost which is 50% of the total architecture (recall section 2.3).

3.2.1. Case study for HSPA

For experimental purpose, we implemented interleavers from the most widely used standard of telecommunication systems: HSPA Evolution [HSP04]. We implemented interleaver used for HSPA and performed experiments for different block lengths and parallelisms.

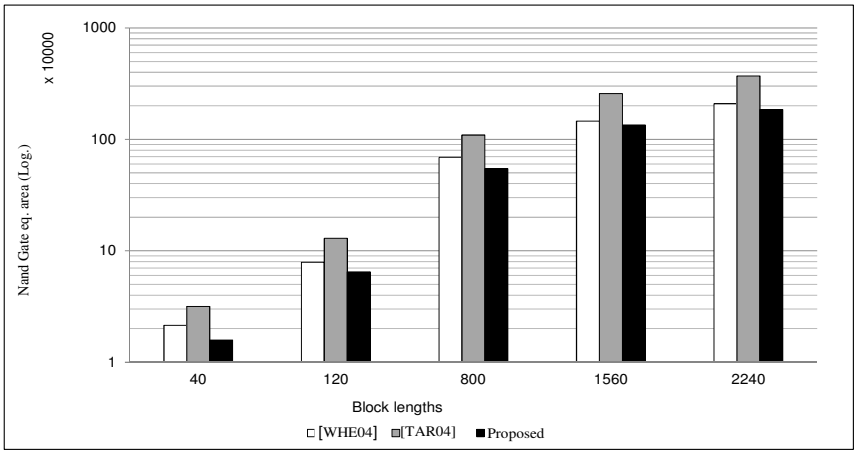


Figure 3. 13 Comparison of HSPA Network Controllers Areas obtained with state of art approaches and Network Relaxation approach

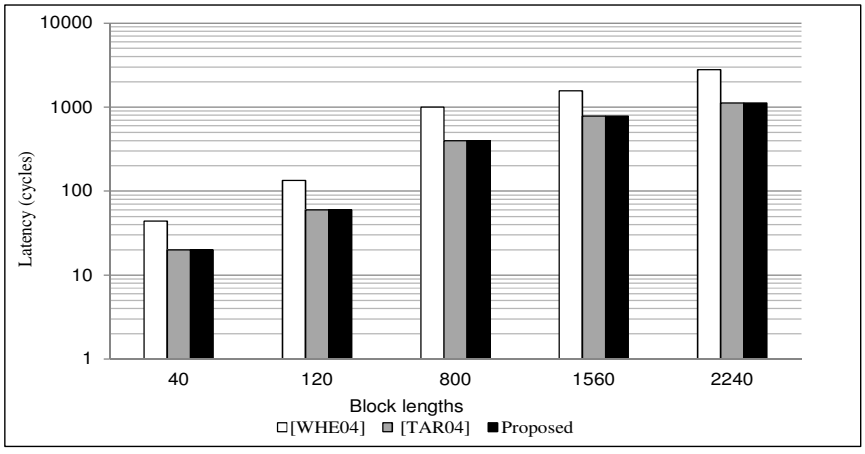


Figure 3. 14 Comparison of HSPA Network Controllers latencies obtained with state of art approaches and Network Relaxation approach

Figure 3. 13 and Figure 3. 14 shows comparison of the proposed network relaxation approach with different state of the art approaches. Different HSPA block lengths *L* with P=4 are considered. The area of the network controller for proposed customized network is compared with the area of the network controller for approaches which uses fully connected (Benes network). Figure 3. 13 shows comparison of area in logarithmical (base10) scale. Our proposed approach has 50% (average) lesser area as compared to [TAR04] and 18% (average) lesser area as compared to [WHE04]. Figure 3. 14 shows latency comparison in term of cycles. The latency show the number of cycles needed to access all the data elements. Our proposed approach has equivalent latency as compared to [TAR04] and 56% (average) lesser latency as compared to [WHE04] (as [WHE04] approach uses FIFOs for conflicting data which is routed in next iteration and as a result the number of cycles increases which increases the latency). Therefore, the proposed approach significantly reduces the cost as compared to [TAR04] with the same latency. On the other hand as compared to [WHE04], our proposed approach reaches small reduction in terms of area, but without any additional latency (i.e. our architecture will achieve higher throughput) leading to a better performance/area tradeoff.

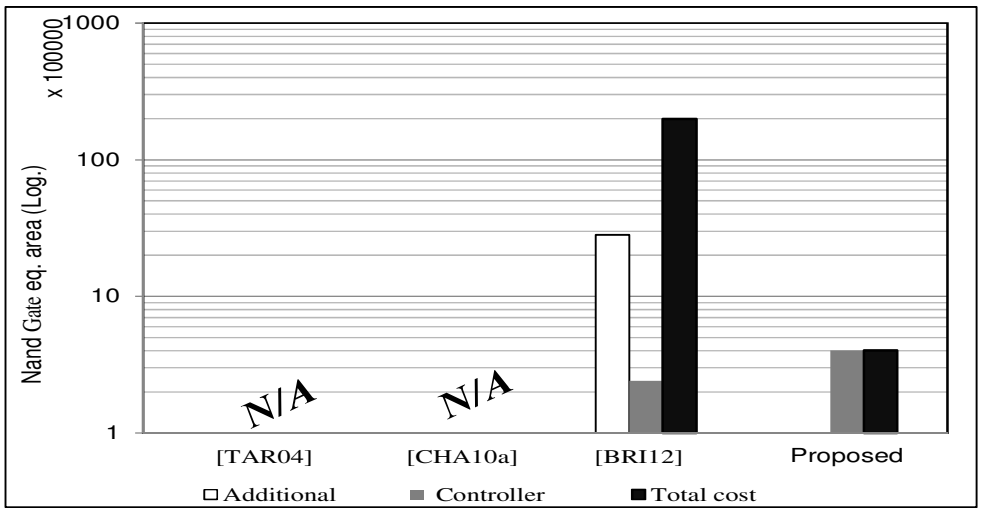


Figure 3. 15. Detailed comparison of HSPA Network Controllers Areas obtained with state of art approaches and Network Relaxation approach
(*L=2240, P=4, Targeted network: Barrel Shifter*)

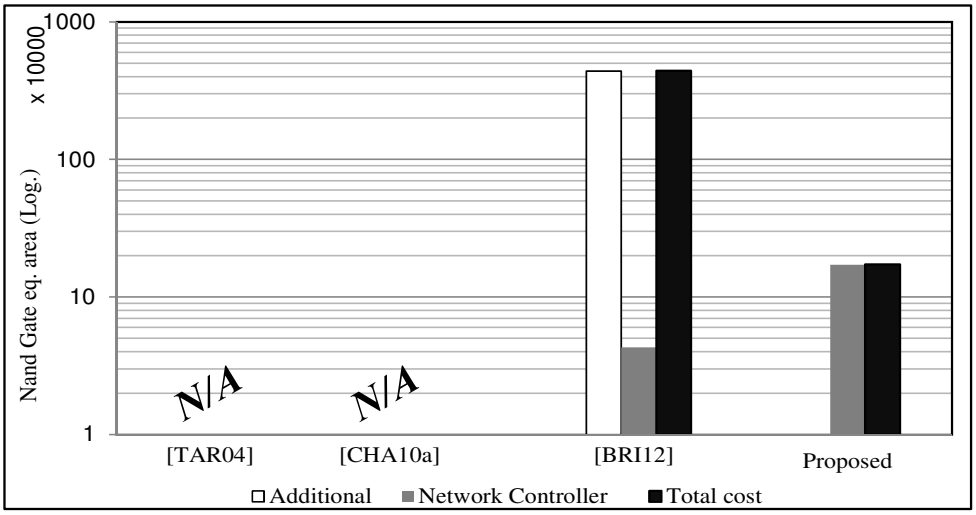


Figure 3. 16 Detailed comparison of HSPA Network Controllers Areas obtained with state of art approaches and Network Relaxation approach
(*L=800, P=8, Targeted network: Barrel Shifter*)

Moreover, we compared our proposed approach with existing approaches [CHA10a], [TAR04] and [BRI12]. These experiments was performed for all block length from HSPA and for parallelism P=4 and P=8. Since the area for the memory banks is the same for each test case, it is not taken into account in these results. Figure 3. 15 and Figure 3. 16 show typical results obtained in this case for 2240 and 800 data and a unique Barrel Shifter BS network constraint. Here, [CHA10a] and [TAR04] are not able to find conflict free memory mapping: architectural constraints are not supported in [TAR04] and it is not possible to find a conflict free memory mapping with a *BS* as network objective for [CHA10a] . However, [BRI12] is able to find the solution with the use of additional registers along with *BS*. As shown in these figures our proposed approach can find an optimized solution with an area divided by about 66 as compared to [BRI12] for the example considered in Figure 3. 15 and by about 20 as compared to [BRI12] for example considered Figure 3. 16. The latency for all the approaches based on memory mapping solutions [TAR04], [CHA10a], [BRI12] and the proposed approach is same as no additional *buffer* elements are needed in the interconnection network.

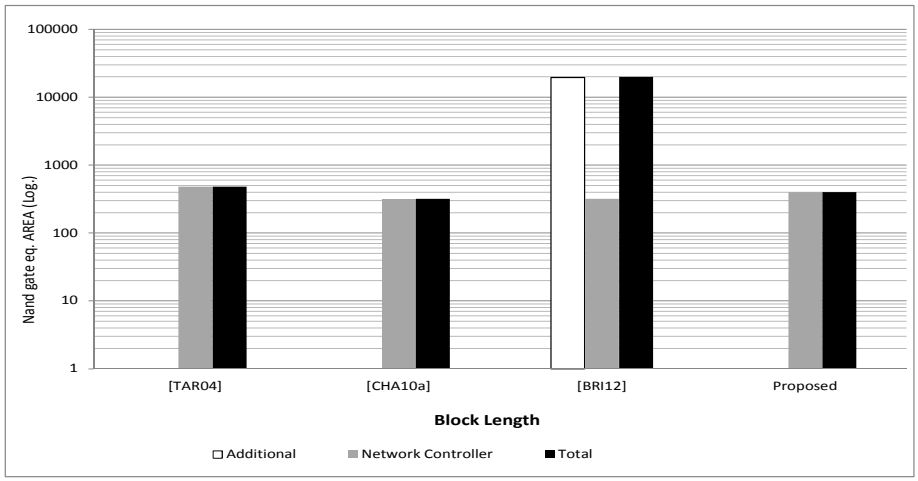


Figure 3. 17. Detailed comparison of HSPA Network Controllers Areas obtained with state of art approaches and Network Relaxation approach
(*L=2240, P=4, No targeted network*)

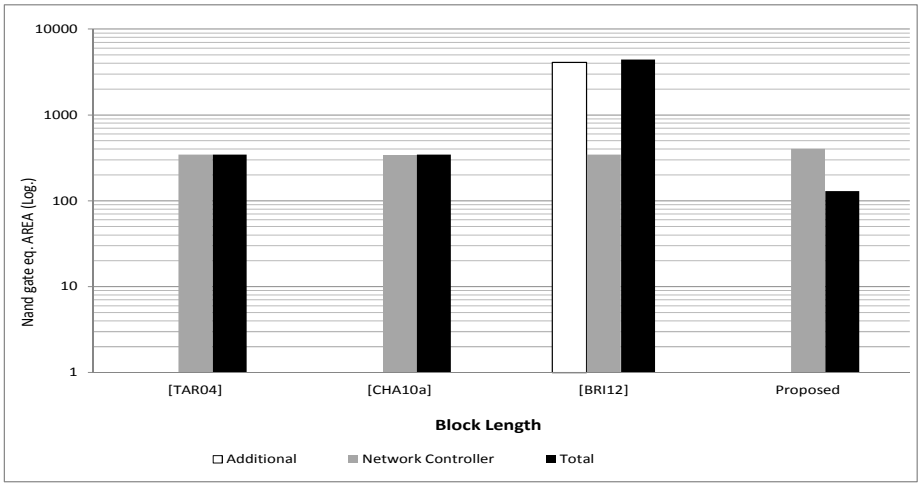


Figure 3. 18 Detailed comparison of HSPA Network Controllers Areas obtained with state of art approaches and Network Relaxation approach
(*L=800, P=8, No targeted network*)

In Figure 3. 17 and Figure 3. 18, the best results for each approach are presented. In this case, the interconnection could be different for each approach, for example [CHA10a] and [TAR04] are not able to find a solution with a Barrel Shifter interconnection network so [TAR04] gives solution only with a fully connected network like Benes and [CHA10a] is able to find a solution with a Butterfly BF network, for L=2240 and P=4. Whereas for the second example with L=800 and P=8 in Figure 3. 18, the minimum cost is achieved with Benes network for [CHA10a][TAR04] . The cost of the [BRI12] is very high due to the addition of buffers and multiplexers to solve the conflicts. Our approach (based on customized network) is able to find the conflict free memory mapping for all the considered examples with an area reduction of 34% in average for Figure 3. 17 and 50% in average in Figure 3. 18.

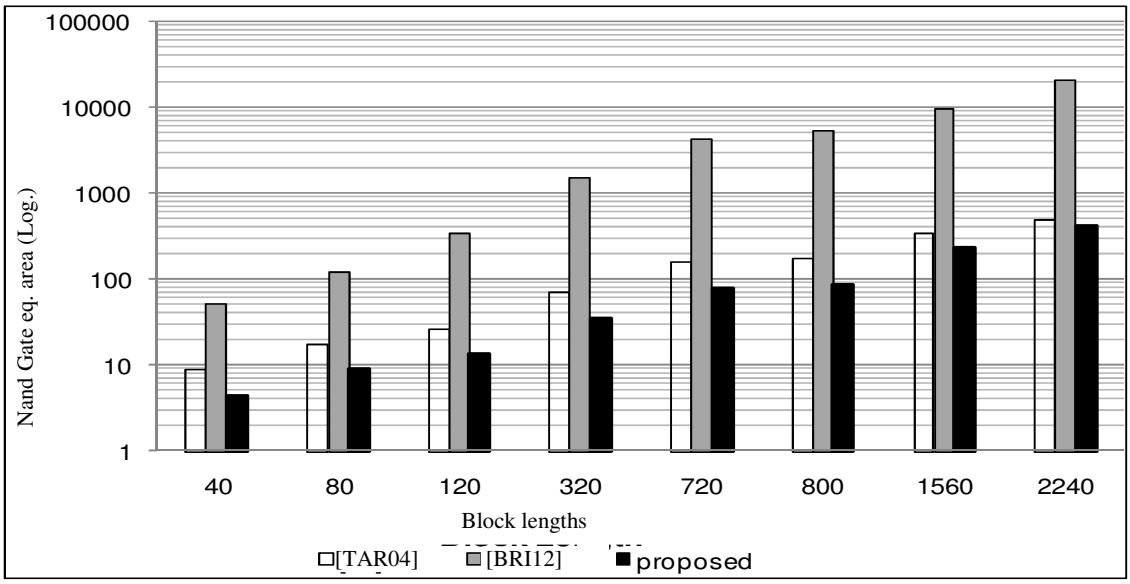


Figure 3. 19. Comparison of HSPA decoder architecture estimated areas obtained with state of art approaches and Network Relaxation approach for different block lengths (*P=4*)

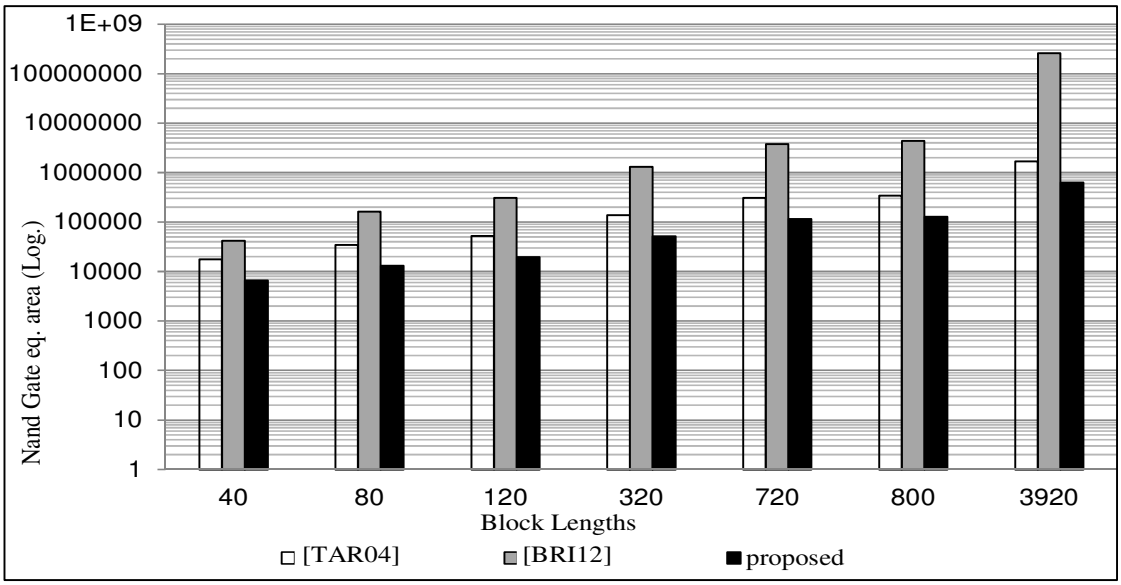


Figure 3. 20. Comparison of HSPA decoder architecture estimated areas obtained with state of art approaches and Network Relaxation approach for different block lengths (*P=8*)

In order to further explore our approach we have considered different block lengths from HSPA with *P=4* and *P=8*. The results for *P=4* is shown in Figure 3. 19 and for *P=8* in Figure 3. 20. Like in the previous experiments, these figures show the best result for each approach. The results clearly show that our proposed network relaxation method always gives lower cost solution as compared to existing approaches. For *P=4*, our solution reduces the total area 21 times in average for all the experiments. Compared to [TAR04] the area is divided by 1.8 in average and compared to [BRI12] the area is divided by 41 in average. For P=8, network relaxation reduces the total area 40 times in average for all the experiments. Compared to

[TAR04] the area is divided by 2.6 in average and compared to [BRI12] the area is divided by 76 in average. As it has been said, the latency is same for all the considered approaches [TAR04][BRI12][CHA10a] and are compared to our proposed approach. On the contrary, in [WHE04] the latency of the architecture is impacted as compared, then this solution is not considered in the final results.

3.2.2. Case study for LTE

We also performed experiments for interleaver used in LTE. In [WON10b], LTE with parallel architecture is presented. They have presented a multistage interconnection network based on barrel shifter. Figure 3. 21 shows the parallel architecture with multistage interconnection network with P = 8 (2^i) with three stage. Configuration bits (bits needed to control the network) for P = 8 can be calculated by considering a stage (3 – i), in which i = 0 to 2, we need one bit for 2^i inputs in that stage (like in first stage we will need 2^0 i-e 1 bit). So, one bit is needed for stage 0, two bits are needed for stage 2 and four bits are needed for stage 3.Therefore, three stages of modified BS need seven bits for configuration at each access. Similarly, for P=4, two stages of modified barrel shifter needs three bits.

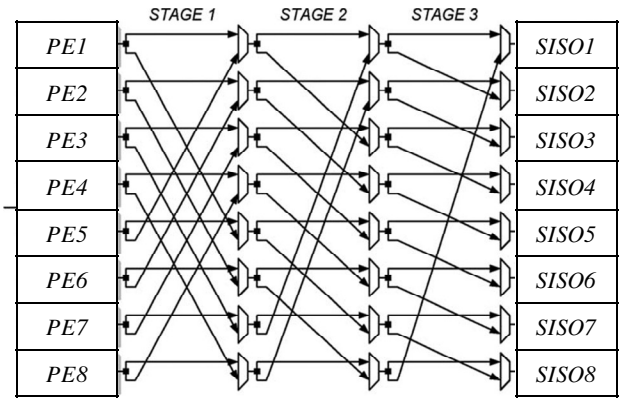


Figure 3. 21. Parallel architecture with multistage network

We have applied our proposed approach for the above mentioned case study. The proposed approach is able to optimize the network as some block lengths can support standard *BS*. Table 3.5 shows *LTE* block lengths that support BS network.

Table 3.5 Block lengths supported by *BS*

P=4	160,200,240,320,360,480,528,800,880,960,1440, 1600,2240,2880,3520,4160,4480,5760
P=8	416,480,800,1600,2240,3520,3584,4608,4480

As a result, the cost of the architecture can be reduced as shown in Figure 3. 22. In this figure, the area comparison between [WON11b] and our approach is given for different block lengths. The proposed approach can save up to 35% of the network area as shown in Figure 3. 22.

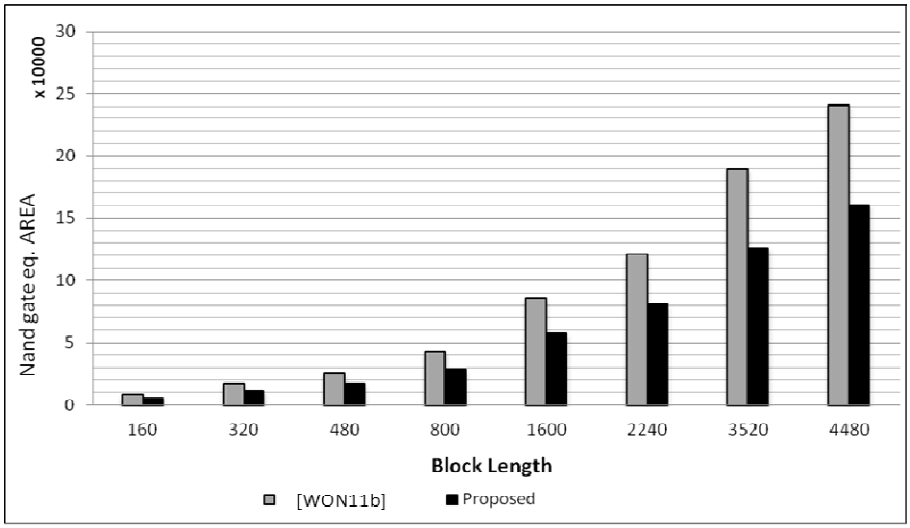


Figure 3. 22. Comparison of LTE decoder architecture estimated areas obtained with state of art approache and Network Relaxation approach for different block lengths (*P=4*)

4. Conclusion

In this chapter, a dedicated approach to explore the design space for parallel turbo decoder architectures is presented in which different configurations based on shuffled and non-shuffled schemes are considered in order to analyze the hardware complexity of the decoder architecture. A novel architecture to decode turbo codes using shuffled scheduling is also proposed in which we have studied and solved different memory issues for shuffled decoding technique. The analysis of the cost estimation gave us the distribution of cost for different parts of the architecture like network controller, memory controller, extrinsic memory. We have seen that network controller is up to 60% of the total cost for some test cases; unfortunately state of art approaches are not able to directly optimize this part. Therefore, we proposed a new approach for conflict free memory mapping based on network customization to generate optimized architectures. This approach also respects targeted network architecture by modifying it with additional network components if needed. Proposed approach is compared through industrial test-cases with the state of the art approaches. Results show that optimized architectures can be obtained by applying proposed network relaxation approach even when particular network is targeted with significant area reduction

and without any reduction in terms of throughput (as the latency of the proposed approach remains the same as state of art memory mapping approaches).

The proposed network relaxation approach provides in-place mapping for conflict problem which can be solved with P banks, and provides *MRMW* mapping for conflict problems which cannot be solved with in-place mapping. However, due to the customized network solution the proposed approach is able to generate optimize solution in both the cases. As mentioned in chapter-2, *MRMW* architecture is more costly in term of area as compared to in-place memory architectures. Therefore, a second option must also be explored in order to optimize the final architecture in term of area: to use in-place memory mapping architecture for other memory conflict problems. This solution is presented in the next chapter.

Chapter 4

IN-PLACE MEMORY MAPPING FOR OPTIMIZED ARCHITECTURE

Table of Contents

In this chapter, we present an optimized parallel hardware architecture using in-place memory mapping. In-place memory mapping architecture and multiple read multiple write (MRMW) architectures are currently used in the state of the art approaches to solve a memory mapping problem. However, the controller MRMW architectures are more costly as compared to in-place architecture. In this chapter, we show that some of the memory mapping problems which are currently solved using MRMW architectures can be solved by using in-place memory mapping architectures. We also present polynomial time algorithms to solve these problems with in-place architecture. As a result the complexity in term of area of the decoder can be reduced.

1. Introduction

In first part of the previous chapter, we have explored different configurations based on shuffled and non-shuffled turbo decoder architecture using state of the art approaches in order to find conflict free memory mapping. Non-shuffled configurations are turbo like problems for which memory mapping approaches based on in-place architecture are used. On the other hand, shuffled configurations represent LDPC like problems for which approaches based on multiple read multiple write (*MRMW*) memory mapping architecture are needed. So, state of the art approaches uses *MRMW* architecture and in-place memory architecture to find conflict free memory mapping. For example, memory conflict problems in non-shuffled decoding of *LTE* [SAC12] and non-shuffled decoding in *HSPA* [SAN13] are solved using in-place memory architecture. While conflict problems solved with *MRMW* architecture includes shuffled decoding in *LTE* [OSC12], shuffled decoding in *HSPA* and non-binary LDPC problem [BRI12] [CHA10b]. The *MRMW* memory mapping architecture requires network configurations for read as well as write operations which double the network controller cost (as explained in chapter 2). However, for in-place architecture the network configuration of a read cycle is also reused for the write cycle. As a result, the network controller is reduced for in-place architecture as compared to *MRMW* architecture.

In this chapter, we introduce that every conflict problem with two accesses (each data element is accessed two times e-g natural order access and interleaved order access) which are currently solved with *MRMW* memory mapping architecture can be solved using in-place architecture. It allows us to reduce the network controller which results in an optimized decoder architecture. Our proposed work addresses the memory mapping problems in which each data is accessed exactly two times.

2. Two access memory mapping problem

In this section we describe memory conflict problems in which each data is accessed exactly two times. These problems include memory mapping problems of turbo decoders in *LTE* and *HSPA* interleaver with shuffled as well as non-shuffled decoding schemes [BER93]. The memory conflict problem in non-binary LDPC is also with two accesses [DAV].In order to find conflict free memory mapping using classical design time approaches, these problems can be categorized as shown in Figure 4.1. The in-place memory mapping solution is possible for conflict problem in UWB, non-shuffled decoding in HSPA and non-shuffled decoding in LTE interleavers whereas the *MRMW* mapping solution is possible for conflict problem in

non-binary LDPC, shuffled decoding in HSPA and shuffled decoding in LTE interleavers. [SAN13] [OSC12] [BRI12] [CHA10b].

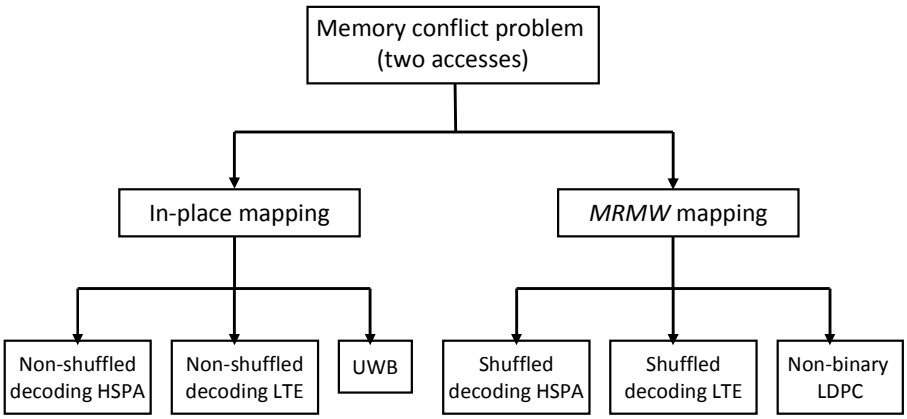


Figure 4.1 Solution for memory conflict problems

In this work, we propose that all of these two access memory conflict problems can be solved by using in-place memory mapping architecture and associated memory mapping algorithms are proposed.

2.1. Problem formulation

Approaches based on in-place memory mapping includes [SAN13] which can solve the given problem in polynomial time using bipartite edge coloring algorithm. Using in-place mapping solution, this approach can only target turbo like problems (e-g non-shuffled HSPA/LTE) in which data is first accessed in natural order and then in interleaved order because of their bipartite nature. However, we can show that it is also possible to use the approach [SAN13] for problems which are not of bipartite nature i-e data elements accessed randomly and not accessed in natural order and interleaved order pattern (like in shuffled *LTE*). This can be done by performing a test which can determine the bipartiteness. This test will be referred as *bipartite test*. After applying the *bipartite test* if the conflict problem results in bipartite graph then we can apply the bipartite edge coloring algorithm to find in-place conflict free memory mapping [SAN13]. Bipartite graph can be defined as below.

Definition ***Bipartite Graph***

Bipartite Graph is a graph whose nodes can be divided into two independent sets, T_x and T_y such that every edge (t_x, t_y) connects a node t_x from T_x to a node t_y from T_y. Moreover, there is no edge that connects nodes of same set.

So the bipartiteness of a graph can be determined by the *bipartite test* which can show whether the graph is bipartite graph or not. Hence, the graph for two access problem can be categorized in bipartite graph and non-bipartite graphs as shown in Figure 4.2. We can find conflict free memory mapping using in-place architecture by applying bipartite edge coloring algorithm in case of bipartite graphs. On the other hand, memory mapping problems in non-binary LDPC codes and shuffled decoding of HSPA interleaver result in a non-bipartite graphs. For these graphs, approaches based on *MRMW* mapping architecture are needed as currently state of the art approaches are not able to find conflict free memory mapping using in-place architecture.

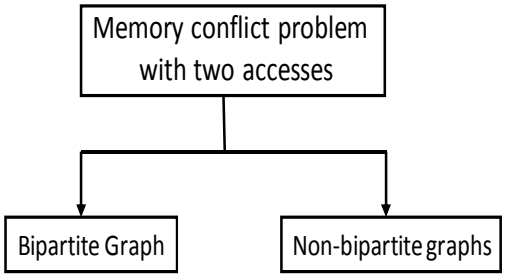


Figure 4.2 Categories in two access problem

In this work, we propose to solve mapping problem using in-place memory mapping architecture even for non-bipartite graphs. We can find memory mapping using Vizing theorem [VIZ64] which can find conflict free memory mapping for any given problem using *P+1* banks. So, thanks to this approach, in-place mapping architecture can be achieved by adding only one additional bank. However, Vizing theorem is true only for simple graphs. We will define simple and multi-graph before introducing Vizing theorem.

Definition ***Simple Graph***

A simple graph is a graph which has no loops (edges connected at both ends to the same node) and no parallel edges (two nodes are connected with only and only one edge).

A simple graph is shown in Figure 4.3 in which no loops and no parallel edges exist.

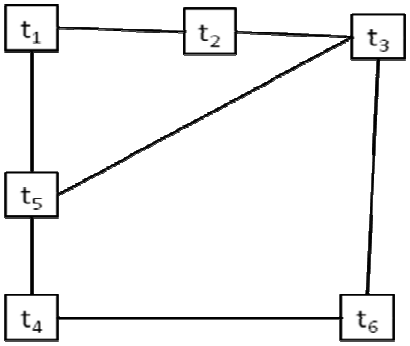


Figure 4.3 Simple graph

Definition ***Multi-Graph***

A multi-graph is a graph which has loops (edges connected at both ends to the same node) and/or parallel edges (two nodes are connected with only and only one edge).

A multi-graph is shown in Figure 4.4 in which a loop exists at node t_4 and parallel edges exist between t_1 and t_3, and between t_6 and t_3.

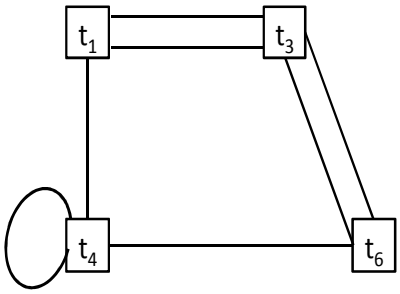


Figure 4.4 Multi-graph

Definition ***Vizing Theorem***

A simple graph of degree P (largest number of edges connected to a node in the graph) *can be edge-colored with at most P+1 colors where P is the maximum degree in a graph* [VIZ64].

Hence, non-bipartite graph for memory mapping problem can be edge colored with *P*+1 colors which results in conflict free memory mapping with *P+1* banks using Vizing theorem. We will perform a test (*simple graph test)* before applying Vizing theorem. For multi-graphs a dedicated approach is used to find conflict free memory mapping.

Finally, if a non-bipartite graph is also a multigraph, then a dedicated approach proposed in this thesis can be used. This proposed approach is based on approach used for transportation problem which can find conflict free memory mapping for any given problem.

The above mentioned proposed work can be formalized by the design flow described in the next section.

3. Design Flow

This entire process is presented in a proposed design flow presented in Figure 4.5. For a given mapping problem first a graph is constructed. Then bipartite test is performed to determine whether the given problem can be solved using *P* banks. If the graph is not bipartite then more than *P* memory banks are needed. Hence, *simple graph test* is performed to check whether Vizing theorem can be applied which can provide conflict free memory mapping

using upto *P+1* banks. Finally, for multi-graphs, a dedicated approach is applied which can find conflict free memory mapping using upto *(3/2)*P* banks.

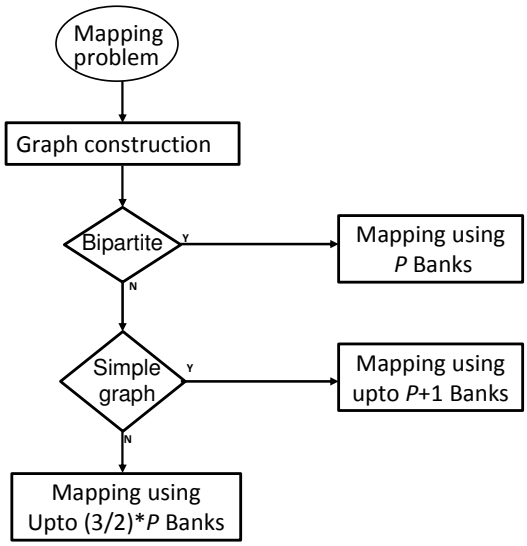


Figure 4.5 Design flow

3.1. Graph construction

The first step of the design flow is to construct a graph based on the memory access matrix. First an initial graph $G' = (T \cup L, E)$ is constructed according to the data access matrix in which set of nodes T represents all the time instances whereas set of nodes L represents all the data elements of the access matrix. An edge (t_i, l_j) is incident to the data node l_j and to the time node t_i if l_j is processed at t_i (i.e. data l_j will be read and next written at time t_i) where $t_i \in T$. Then the graph G' is converted into a graph $G = (T, E)$.

The two access memory mapping problems have the following two distinct properties:

Property 1

The number of parallel accesses to data the elements P (i.e. number of data required to access concurrently) at any time instance is always same. This property implies that in G', each time node has same degree, $d_t = P$.

Property 2

Each data element is accessed exactly two times. This property implies that all the data nodes have the same degree, $d_l = 2$.

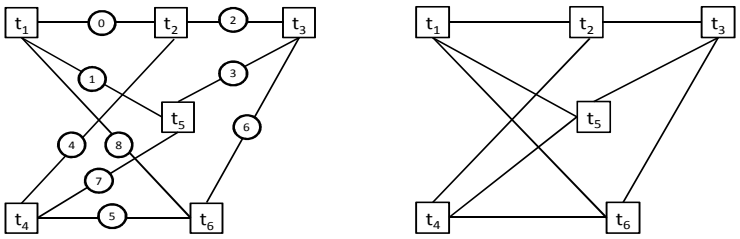
According to the property 2, the graph *G'* can be converted into graph *G* by first joining two edges at each data node and then removing all the data nodes from G_i. Thanks to property 1, *G* is regular i-e all nodes have same degree with the degree of each time node, $d_t = P$. Now

we can conclude that in set of *T,* each edge t_i corresponds to a data node in G_i, so the coloring of edges in *G* actually means coloring of data nodes in G_i.

PE_1	0	0	2	4	1	5
PE_2	1	2	3	5	3	6
PE_3	8	4	6	7	7	8
	t_1	t_2	t_3	t_4	t_5	t_6

Figure 4.6 Example of data access matrix with two accesses to each data

Consider a simple example of data access matrix (see Figure 4.6) in which size of data elements *L = 9, parallelism P=3* and number of time instances *T = 6*. For this data access matrix, the graph G_i is shown in Figure 4.7(a) in which set of data nodes is *L = 9* and set of time nodes is *T = 6*. This graph is then converted into the graph *G* shown in Figure 4.7(b). Now we can get a conflict free memory mapping by applying edge coloring algorithms where each edge connected to a time instance will have different colors which means that each data accessed at a time instance is placed in different memory banks resulting in a conflict free access to the data.



(a) Graph G_i (b) Graph *G*

Figure 4.7 General Graph for edge coloring

The graph *G* is needed to be edge colored in order to find conflict free memory mapping. However, for approach proposed in [SAN13] a bipartite graph is required for edge coloring. Therefore, the bipartiteness of the graph is checked by *bipartite test* as described in next section.

3.2. Bipartite test

Bipartiteness of a graph can be tested by using *breadth first search algorithm* (*BFS*). The *BFS* begins at a source (starting) node and inspects all its neighboring nodes. Then for each of those neighboring nodes (one after another), their own neighbor nodes which have not been visited before are traversed, and so on. We start the search at any node and assign it to alternating sets i-e assign the starting node to set-1 and assign its entire connected neighbor nodes to set-2, further assign to set-1the nodes connected to the all neighbors of the neighbors and so on. The graph is not bipartite if at any step a node has connected neighbors assigned

with the same set. In the end, the graph is bipartite, if the search ends without having two connected neighbors assigned to the same set.

As shown in Figure 4.8, the algorithm uses a queue data structure to store intermediate results as it traverses the graph. A queue is a collection in which the entities are kept in order as same as First-In-First-Out (FIFO). The addition of entities is made to the rear terminal position, known as en-queue, and removal of entities is made from the front terminal position, known as de-queue.

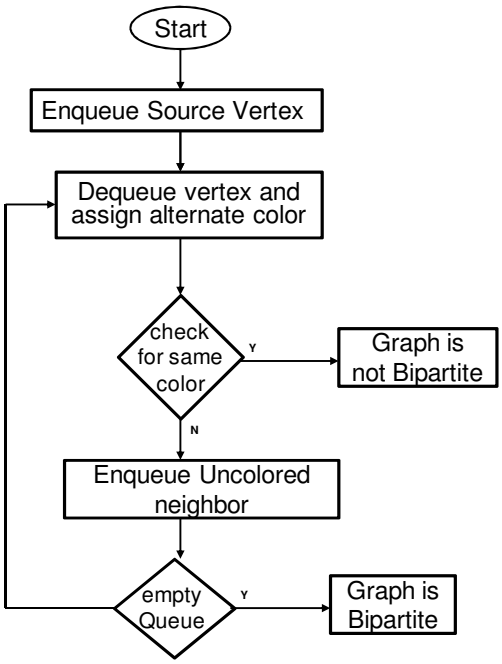


Figure 4.8 Bipartite test

The whole process is shown in the following steps.

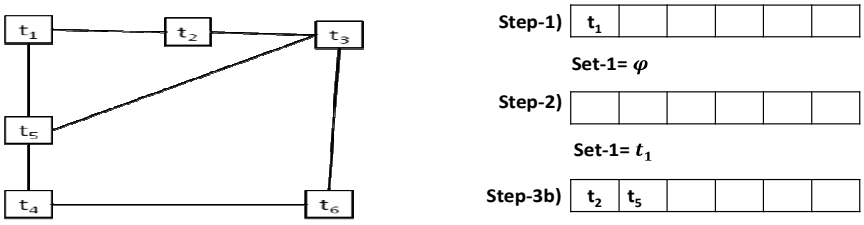
1) The first step is to enqueue the source node as shown in Figure 4.9(b) in which t_1 is considered as the source node in the graph shown in Figure 4.9(a).
2) Dequeue a node and assign to a set.

3a). If two adjacent node are assigned to same set then the graph is not a Bipartite graph then the algorithm is completed

3b). Otherwise enqueue the neighbor nodes (the direct child node) which are not yet assigned.

As it is shown in Figure 4.9(b), t_1 is dequeued and assign it to set-1. Then all the neighbours nodes t_2 *and* t_5 are enqueued.

4) If the queue is empty, every node on the graph has been assigned then the algorithm is completed, else repeat previous step by assigning alternate sets to the neighbors.



(a) (b)

Figure 4.9 Birpartite test example

As a result, the bipartiteness of a graph can be decided for a conflict problem, so bipartite edge coloring approach can be used to find conflict free memory mapping [SAN13]. However, for non-bipartite graph we can apply the Vizing theorem in case of simple graphs (see Figure 4.5). The *simple graph test* is described below.

3.3. Simple graph test

Simple graph has two properties: no loops and no parallel edges. The access of one data element twice at a time instance always results in a graph for memory mapping problem with loops which is never the case for memory access problems. Therefore, we can easily perform a test to determine whether a graph is a simple graph or not by checking the existence of parallel edges in the graph by modifying the breadth first algorithm.

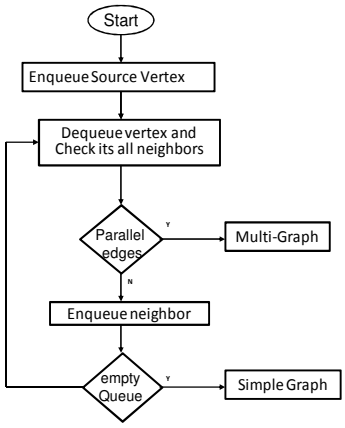


Figure 4.10 Simple Graph test

As shown in Figure 4.10, the algorithm traverses the graph as follows:

1) Enqueue the source node (see Figure 4.11).

2) Dequeue a node and check all its neighbours (see Figure 4.11).

3a) If there exist a parallel edge between the neighbor and the dequeued node.

3b) Otherwise enqueue the neighbor nodes (the direct child node) which are not yet traversed as shown in Figure 4.11.

4) If the queue is empty, every node on the graph has been visited then the algorithm is completed, else repeat previous step.

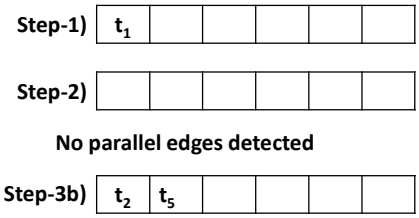


Figure 4.11 Simple graph test example

We start traversing the graph by checking whether there exists a parallel edge. As soon as a parallel edge is detected, we stop traversing the graph and conclude that the graph is a multigraph and we cannot apply Vizing theorem. On the other hand if no parallel edges are detected in the entire graph, then it is a simple graph and we can apply Vizing theorem which can resolve the memory conflict problem by using *P+1* banks.

3.4. Vizing theorem for edge coloring

The process of edge coloring using Vizing theorem is shown in Figure 4.12 in which first a node is selected to color. Then vizing fan (defined later in this section) is constructed and if same colors are not available at both nodes of the edge. The fan is ended in two possible cases: 1) without any repetition of missing colors (the un-used colors) at the nodes of the fan, 2) with repetition of missing colors at the nodes of the fan. In the second case, first path is traced and then existence of the loop in the path is detected. So a path is selected based on the loop detection. Finally, the colors are exchanged and the final coloring is assigned to the graph. As a result, the desired edge is colored. This whole process is described below in details and an example is provided in the next sub-section.

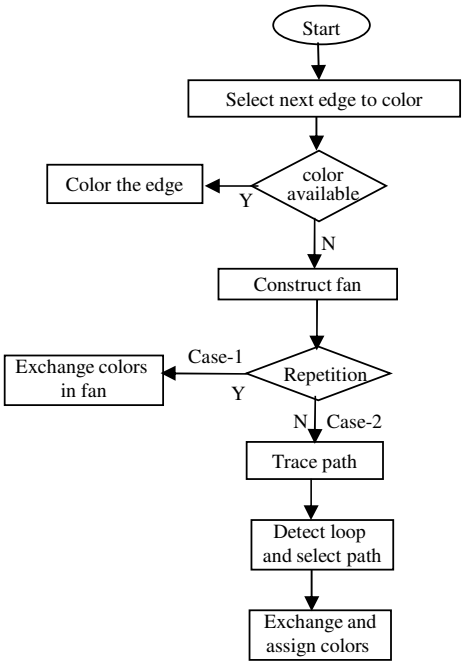


Figure 4.12 Edge coloring flow based Vizing theorem (up to *P+1* colors)

We have to color the edges of graph G by using at most $P+1$ colors. Let us consider a graph G having $j+1$ time nodes with a maximum degree P. Edges of G are $e_1, e_2 \ldots e_j$. We will start from an empty graph G_0 with $j+1$ time nodes without any edge. This graph is then extended to the graph $\mathbf{G}_1$ in which one edge (e_1) is considered. We keep on adding edges to color them until the whole the graph is completed. Mathematically, in each iteration we extend the graph coloring of $\boldsymbol{G_{i-1}}$ to colour $\boldsymbol{G_i = G_{i-1} \cup \{e_i\}}$, **For** $\boldsymbol{i}$ = *1 to j*.

We explain how to color $\boldsymbol{G_i}$ using at most $P+1$ colors. Inductively, we suppose that we have already colored the edges of $\boldsymbol{G_{i-1}}$ using at most $P+1$colors.

Now $\boldsymbol{G_i = G_{i-1} \cup \{e_i\}}$ where $\boldsymbol{e_i = (t_0, t_1)}$ be the next edge to color (*conflict edge*) as shown in Figure 4.13. Mathematically, in each iteration we extend the graph coloring of $\boldsymbol{G_{i-1}}$ to colour $\boldsymbol{G_i = G_{i-1} \cup \{e_i\}}$, **For** $\boldsymbol{i}$ = *1 to j*. As shown in Figure 4.13 at most $P - 1$ edges containing $\boldsymbol{t_0}$ or $\boldsymbol{t_1}$ are colored. If the missing color at $\boldsymbol{t_0}$ is same as missing color at $\boldsymbol{t_1}$ then color the edge $(\boldsymbol{t_0, t_1})$ with that missing color. Otherwise if the missing color at $\boldsymbol{t_0}$ is not the same as missing color at $\boldsymbol{t_1}$ i-e color $\boldsymbol{c_0}$ is missing at $\boldsymbol{t_0}$ and one color $\boldsymbol{c_1}$ is missing at $\boldsymbol{t_1}$, then we have no simple solution to color the *conflict edge*.

In order to find solution to color the *conflict edge* we will first construct a sequence known as *Vizing fan*. We have represented each node notation $\boldsymbol{t_i(c_i)}$ in which $\boldsymbol{t_i}$ is the time node and $\boldsymbol{c_i}$ is the missing color available for coloring at that code.

Vizing fan

A fan is an ordered sequence of edges at a node t_0 such that the other connected node t_x has a missing color c_x which is the color of the next edge in the sequence of connected nodes with t_0.

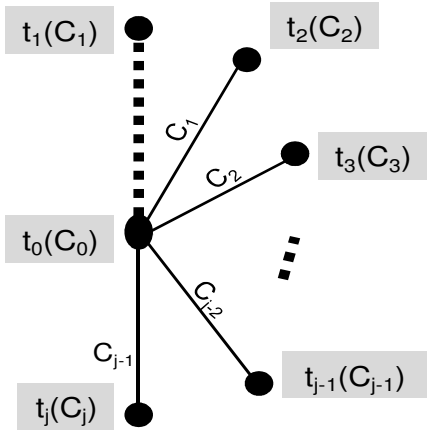


Figure 4.13 Vizing fan

Vizing *fan* can be expressed by constructing a sequence of distinct colors $\boldsymbol{c_1, c_2, \ldots, c_{j-1}, c_j}$ and a sequence of edges $(\boldsymbol{t_0, t_1})$, $(\boldsymbol{t_0, t_2})$, . . . , $(\boldsymbol{t_0, t_j})$. This sequence can be expressed

mathematically as color c_i is missing at t_i for $0 \leq i \leq j$ and color c_i is the color of the edge (t_0, t_{i+1}) for $1 \leq i \leq j$ as shown in Figure 4.13. We can see that color c_j is a missing color at t_j which is available for coloring. If there exists an edge (t_0, t_{j+1}) colored with c_j, where t_{j+}1does not belongs to set $\{t_1, \ldots, t_j\}$, then continue constructing the sequence with the defined c_j and t_{j+1}. Since t_0 has only P neighbors, the construction process completes with one of the following cases:

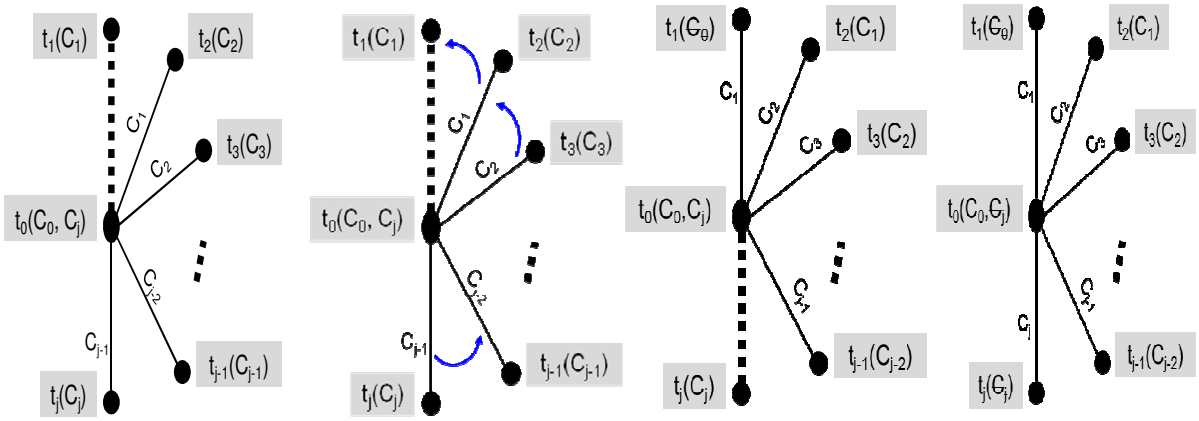
Case-1: No repetition of missing color at the nodes of the fan. While constructing the sequence of the fan, there exist no edge (t_0, t_i) colored with c_j for $1 \leq i < j$.

Case-2: Repetition of missing color at the nodes of the fan. While constructing the sequence of the fan, there exist a color $c_j = c_{k-1}$, for some $2 \leq k < j$: such that the edge (t_0, t_k) is colored with c_j.

Case-1

In this case, the construction of the sequence is completed in such a way that the colors of all the edges of the sequence are not similar to the missing color of the last node. Mathematically, we can say that there exist no edge (t_0, t_i) colored with c_j for $1 \leq i < j$.

In this case, it will always happen that the color missing at the starting node of the *conflict edge* will be also missing at the last node of the sequence i-e the color c_j will be missing at t_0 and t_j (see Figure 4.14(a)). In order to color the *conflict edge* (t_0, t_1), first of all we will shift the *conflict edge* to the last edge of the sequence. The shifting is done by exchanging colors in a sequence that we keep on exchanging the *conflict edge* with its neighboring edge until it is transferred to the last desired edge i-e shift colors (t_0, t_i) with c_i for $1 \leq i \leq j-1$ as shown in Figure 4.14(b-c). In these figures the *conflict edge* (t_0, t_1) is first exchanged with its neighboring edge (t_0, t_2). As a result, the color of (t_0, t_1) becomes c_1 and the *conflict edge* is transferred to (t_0, t_2). Now we exchange the *conflict edge* (t_0, t_2) with its neighboring edge (t_0, t_3). As a result, the color of (t_0, t_2) becomes c_2 and the conflict edge is transferred to (t_0, t_3). We keep on exchanging the *conflict edge* with its neighboring edge and at last (t_0, t_j) will become *conflict edge* (see Figure 4.14(c)). It can be seen that c_j is missing at both t_0 and t_j so we can color (t_0, t_j) with c_j (see Figure 4.14(c)) and hence the *conflict edge* is colored.



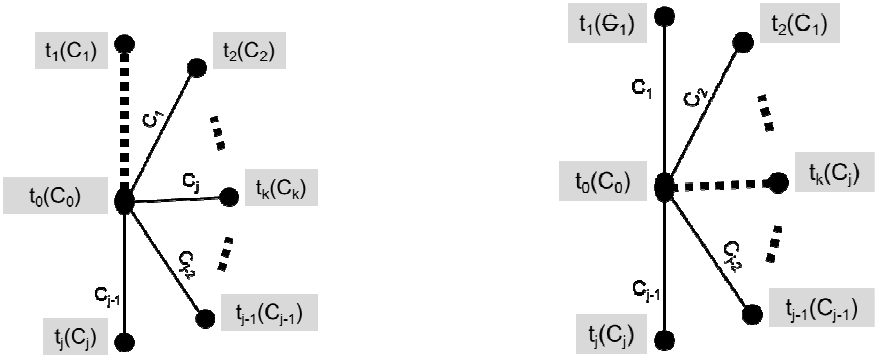
(a) (b) (c) (d)

Figure 4.14 Vizing theorem Case-1

Case-2

In case-2, the construction of the sequence is completed in such a way that there exists one color in all the edges of the sequence which is similar to the missing color of the last node. Mathematically, we can say that there exist a color $c_j = c_{k-1}$, for some $2 \leq k < j$, such that the edge (t_0, t_k) is colored with c_j as shown in Figure 4.15(a).

In this case, first of all we have to shift the *conflict edge* from (t_0, t_1) to (t_0, t_k) i-e to shift color (t_0, t_i) with c_i for $1 \leq i \leq k-1$. The edge (t_0, t_k) becomes the *conflict edge* and c_j is missing at both t_k and t_j as shown in Figure 4.15(b).



(a) (b)

Figure 4.15 Vizing theorem Case-2

In order to color the *conflict edge,* we have two further sub-cases which have different conditions based on the nodes t_0, t_k, t_j as they can be connected to each other or disconnected. So we check these cases in order to color the *conflict edge*.The two sub-cases are:

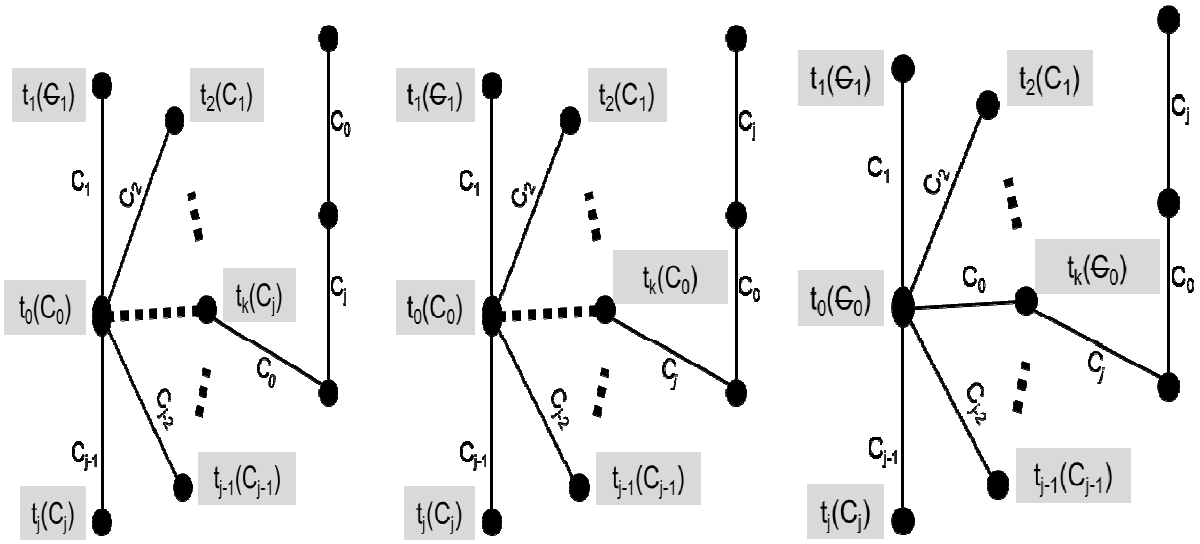
Case-2.a: No loop in the traced path. t_0 and t_k are in different connected components in the graph.

Case-2.b: Loop in the traced path. t_0 and t_j are in different connected components in the graph.

Case-2a

In this case, t_0 and t_k are in different connected components (see Figure 4.16(a)) as the path from t_0 is not ending at t_k. So the *conflict edge* can be colored as follows.

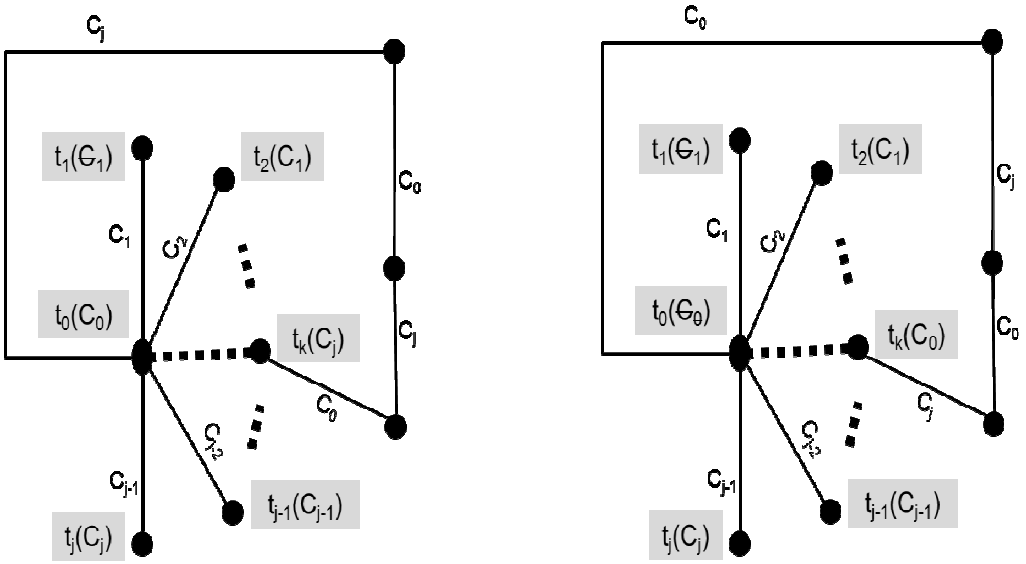
The first step traces path and then exchanges colors. We trace the c_0, c_j path from t_k i-e to find the other node connected with c_0 and search for c_j at that node and then c_0 at the next node and so on as shown in Figure 4.16(a). Then we exchange the color c_0 with c_j in the path as shown in Figure 4.16(b) in which c_0 is replaced with c_j and c_j is replaced with c_0. As a result, c_0 will become missing at both nodes t_0 and t_k. Now, we can color (t_0, t_k) with c_0 (see Figure 4.16(c)). Hence, the *conflict edge* is colored.



(a) (b) (c)

Figure 4.16 Vizing theorem Case-2a

However, if t_0 and t_k are in connected components i-e the path from t_0 is ending at t_k as shown in Figure 4.17(a) then by applying the above procedure, the exchange of color c_0 with c_j will make c_0 available at t_k but it will not be available at t_0 as shown in Figure 4.17(b). So we have no solution in this case. So *case-2b* will be followed to color the *conflict edge*.

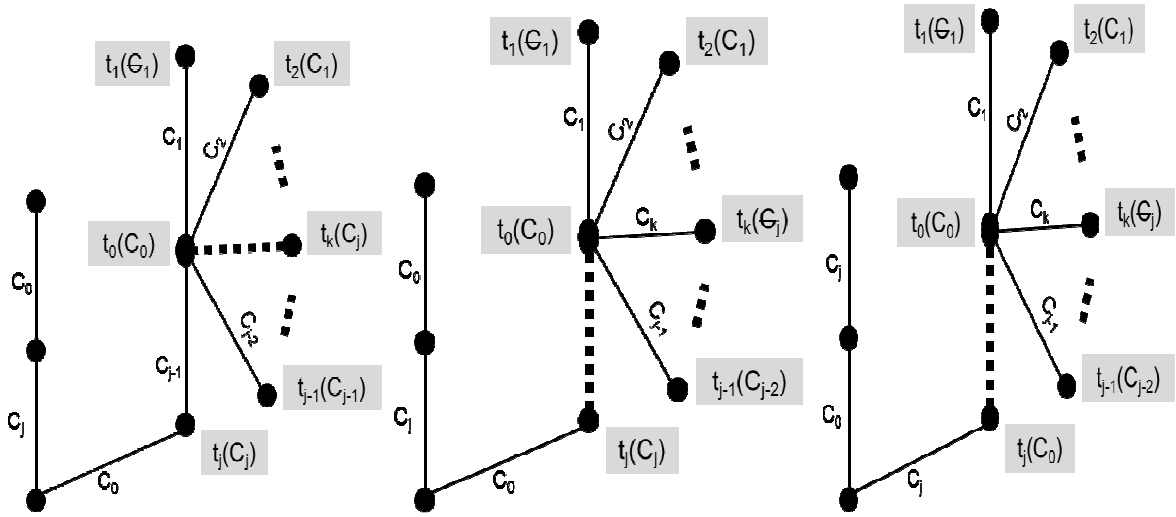


(a) (b)

Figure 4.17 Vizing theorem Condition for Case-2b

Case-2b

In this case, t_0, t_j are different connected components as t_0 is not ending at t_j, so first the path c_0 and c_j is traced at t_j as shown in Figure 4.18(a). Then we shift *conflict edge* from t_k to t_j such that (t_0, t_j) will become *conflict edge* (see Figure 4.18(b)).



(a) (b) (c)

Figure 4.18 Vizing theorem Case-2b

Then, we exchange colors between c_0 and c_j in the t_j -path as shown in see Figure 4.18(c). This will make c_0 missing at t_0 and t_j. So we can color (t_0, t_j) with c_0 as shown in Figure 4.19. Hence, the *conflict edge* is colored.

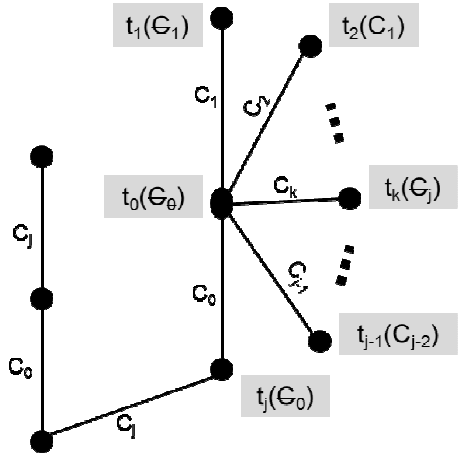


Figure 4.19 Vizing theorem Case-2b final

Complexity of the algorithm

The complexity of the algorithm using Vizing theorem can be calculated by considering a graph with *m* edges to color with *n* time nodes. At each time node we have *P* edges. The complexity to draw a *fan* is O(P^2) as we have to examine each edge at the missing color at that edge. O(*n*) is the complexity to exchange colors for coloring an edge. So overall complexity for the complete graph is O($n\,m + P^2\,m$).

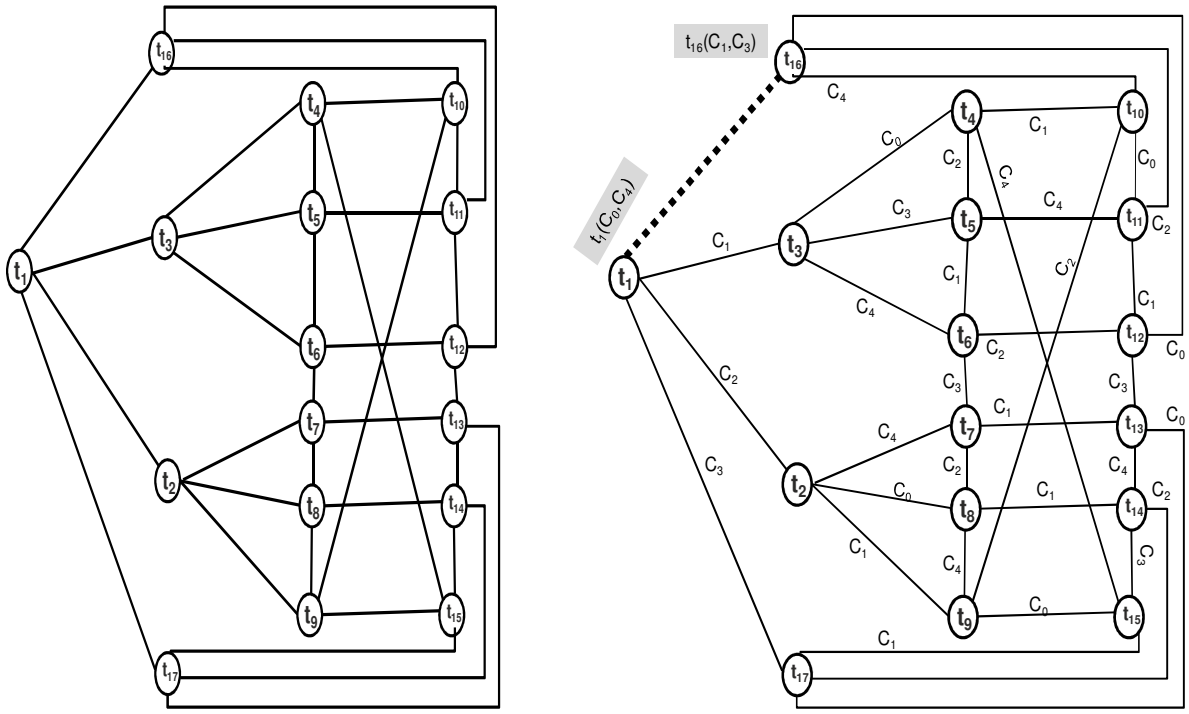
3.4.1. Pedagogical Example

We present an example to explain the Vizing theorem. The data access matrix for the considered example is shown in Figure 4.20. The graph is shown in Figure 4.21(a). As $P = 4$ so we have five colors (4+1) to color this graph.

PE_1	0	2	1	18	16	13	4	5	6	21	31	31	12	10	9	23	3
PE_2	1	4	14	19	17	14	11	7	7	22	33	25	26	32	20	24	29
PE_3	2	5	17	20	19	15	12	10	8	8	22	15	27	28	30	25	28
PE_4	3	6	18	21	33	16	13	11	9	23	24	26	32	30	29	0	27
	t_1	t_2	t_3	t_4	t_5	t_6	T_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	T_{14}	t_{15}	t_{16}	t_{17}

Figure 4.20 Example for Vizing theorem

Figure 4.21(b) shows that we have already partially colored the edges of G_{i-1} using P+1colors. The five colors shown in the figures are c_0, c_1, c_2, c_3, and c_4 for this example. The dotted line represents *conflict edge* e_i to be colored. Colors c_0 and c_4 are missing at t_1 and c_1 and c_3 are missing at t_{16} (see Figure 4.21(b)).



(a) Graph without coloring (b) Graph with partial coloring

Figure 4.21 Graph for Figure 4.6

The fan is shown in Figure 4.22(a) in which the color missing at t_{16} is $\boldsymbol{c_1}$ (consider only one) so we choose the next edge of the fan which is colored with $\boldsymbol{c_1}$. The other time instance connected to edge with color $\boldsymbol{c_1}$ is t_3 in which $\boldsymbol{c_2}$ is missing so the next edge of the fan should be of color $\boldsymbol{c_2}$ and so on. The fan will end with the following two cases:

Case-1: when there is no repetition of missing colors at the nodes of the fan.

Case-2: when there is a repetition of missing colors at the node of the fan.

Case-1

The fan will end without any repetition in the missing colors having *P* edges as shown in Figure 4.22(a) where all the missing colors ($\boldsymbol{c_1}$ and $\boldsymbol{c_3}$ at t_{16}, $\boldsymbol{c_2}$ at t_3, $\boldsymbol{c_3}$ at t_2, $\boldsymbol{c_4}$ at t_{17}) are not repeated at next nodes as missing colors. In this case, there must be a color missing at last node of the fan t_{17} is also missing at starting node t_1. The *conflict edge* can be colored as:

- As the missing color at t_1 is missing at t_{17} so shift the dotted line edge to the t_{17} as shown in Figure 4.22(b)-(c). The *conflict edge* is transferred to t_3, t_2, and then to t_{17}.

- Now we can color the *conflict edge* with $\boldsymbol{c_4}$ as it is missing at both t_1 and t_{17} as shown Figure 4.22(d).

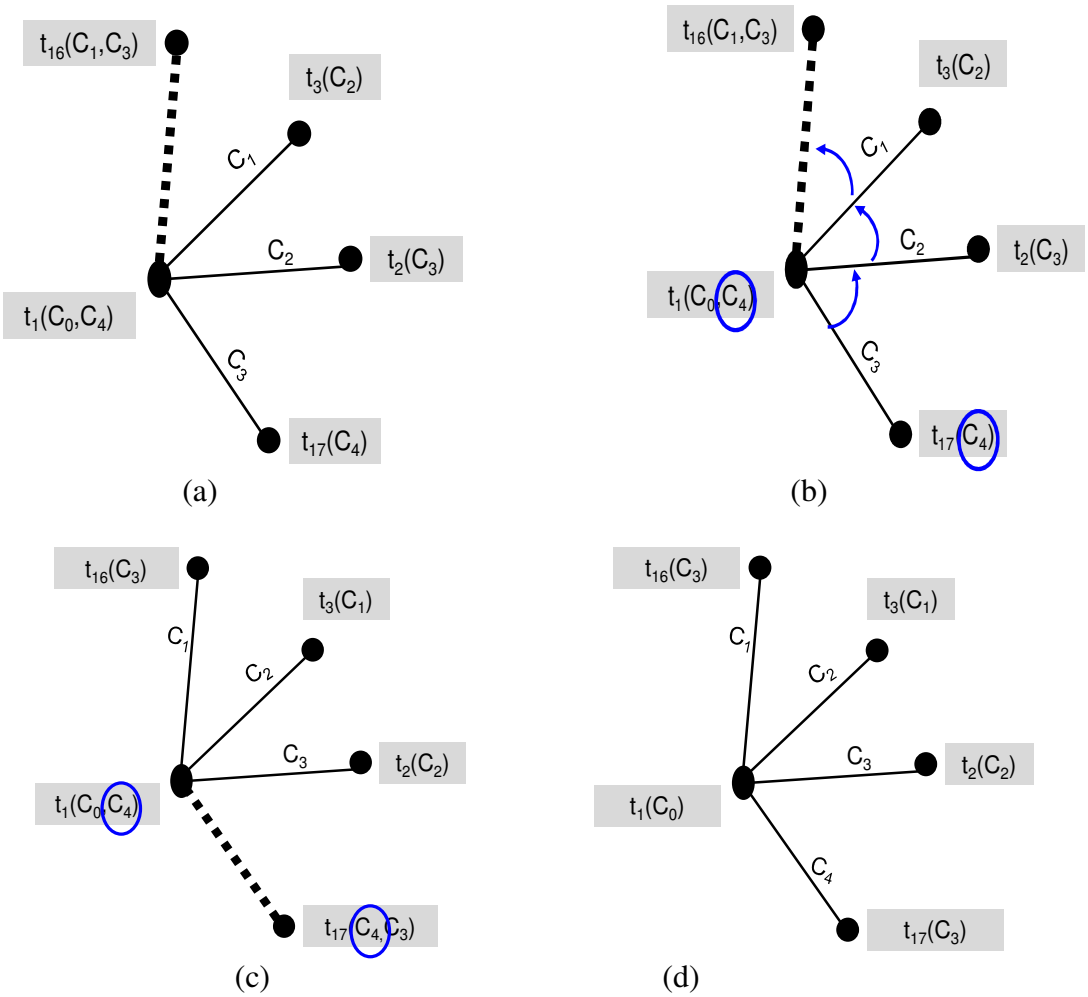


Figure 4.22 Vizing theorem Case-1

Case-2

In case-2 we start constructing the fan and the construction of the fan is ended as soon as a repetition of the missing colors is detected in the fan as shown in Figure 4.23(a). If there is any remaining edge(s) connected to t_1 we will not consider it in the fan. Afterwards, transfer the *conflict edge* to the edge colored $\boldsymbol{c_1}$ (color which has repetition).

Now at t_3, trace the path of $\boldsymbol{c_0}$ (missing color at t_1) and $\boldsymbol{c_1}$ (color of repetition) as shown in Figure 4.23(b). At this stage, there could be further two sub-cases:

Case-2a: t_3 is not making a loop with t_1.

Case-2b: t_3 is making a loop with t_1.

Case-2a

First of all we will check if the node t_3 not making a loop with the node t_1. If it is the case then first we trace the $\boldsymbol{c_1}$, $\boldsymbol{c_0}$ path and then exhange the color $\boldsymbol{c_1}$ and $\boldsymbol{c_0}$ with each other in the traced path. As a result $\boldsymbol{c_0}$ will become missing at t_3 as shown in Figure 4.23(c). Now color

the *conflict edge* with c_0 as it is aviable at both of the nodes t_1 and t_3 as shown in Figure 4.23(d).

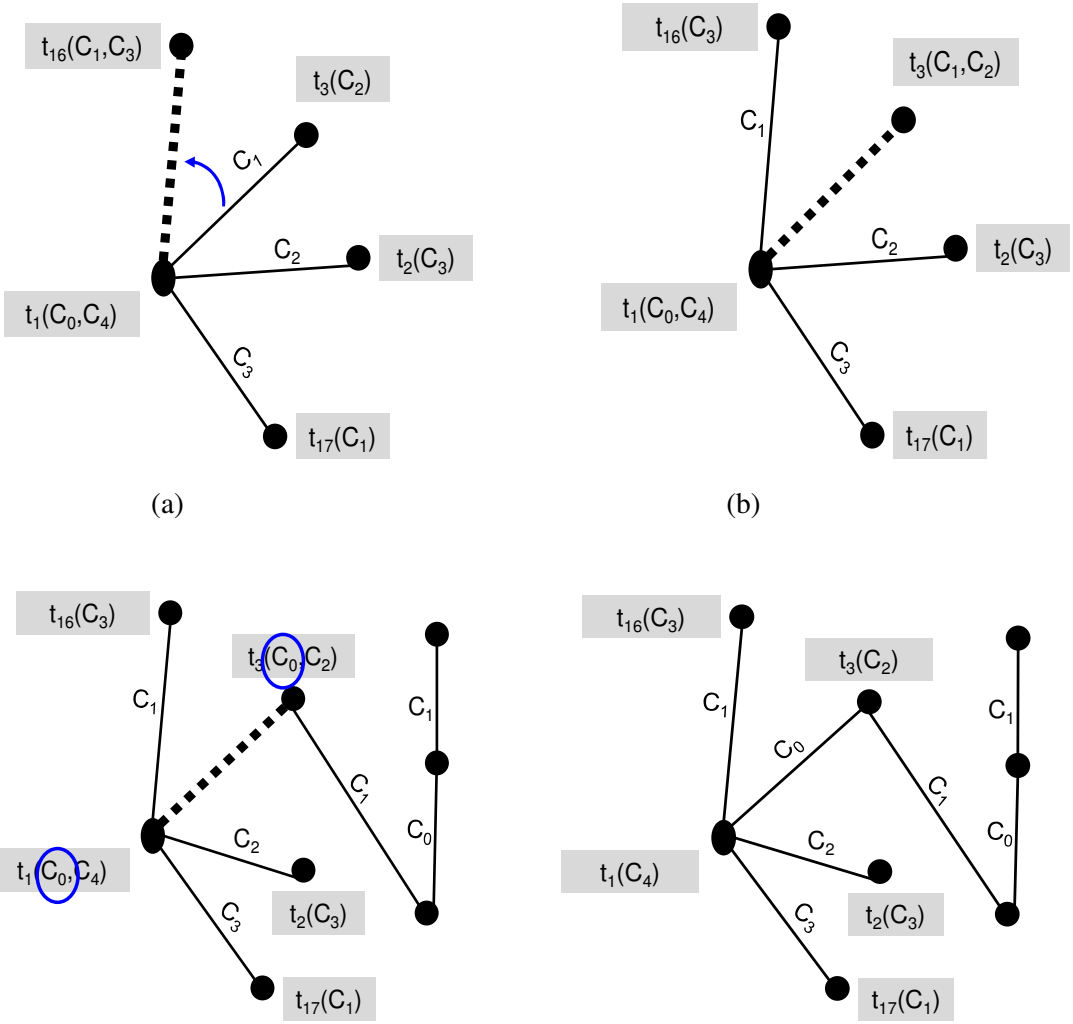


Figure 4.23 Vizing theorem Case-2a

Case-2b

If the node t_3 is making a loop with t_1 then by exchanging colors c_0 will become missing at t_3 but c_0 will not be a missing color at node t_1 so the conflict edge cannot be colored with c_0 as shown in the Figure 4.24(a) and (b). Therefore, in this case we will not exchange colors at node t_3.

If the node t_3 is making a loop with t_1 then the node t_{17} will never make a loop with t_1. Therefore, first of all we will transfer the *conflict edge* to the t_{17} (time instance with repetition of c_1) and trace the path of c_0 and c_1 as shown in Figure 4.24(c) in which there is only one c_0 edge in this path.

Then we will exhange c_1 and c_0 with each other which will make c_0 avialable at t_1 and t_{17} so we can color the *conflict edge* with c_0 as shown in Figure 4.24(d).

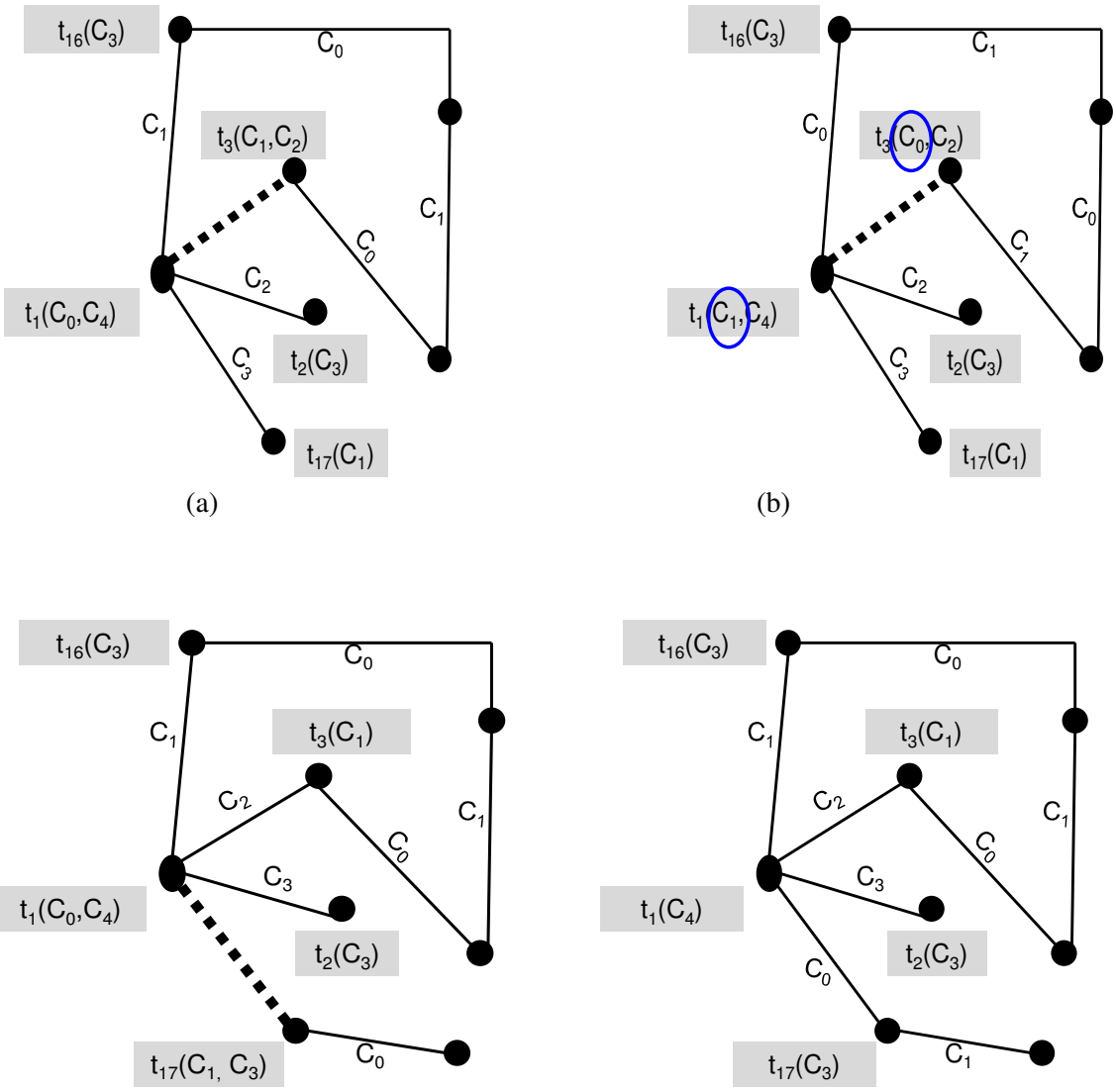


Figure 4.24 Vizing theorem Case-2b

The resultant architecture is shown in Figure 4.25 which represents the final mapping using five banks (b_0, b_1, b_2, b_3, b_4) for the considered example (as each color represents a bank). It can be seen that we need one additional bank for the conflict free mapping but the resultant architecture is based on in-place which can reduce the network controller up to 50% as compared to *MRMW* architecture.

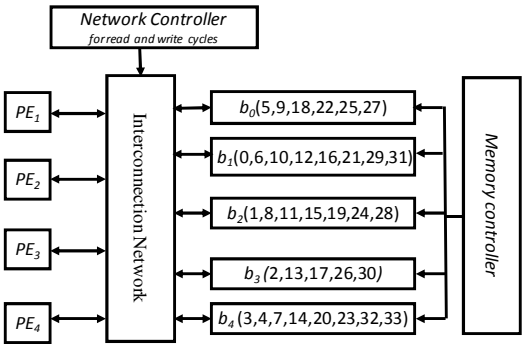


Figure 4.25 Resultant in-place architecture

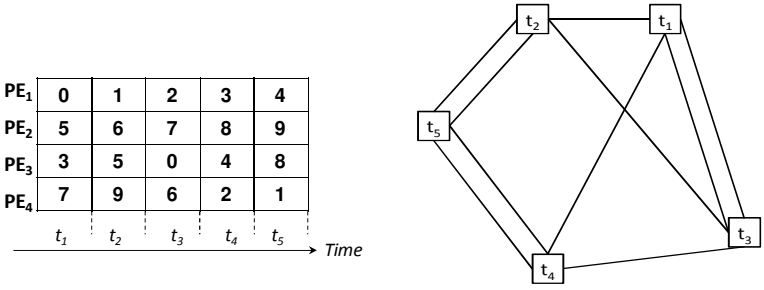
3.5. In-place memory mapping for multigraphs

In the previous section, we have shown that Vizing theorem can be applied only for simple graphs as shown in Figure 4.5. The proposed approach for multi-graphs is described here in details

A dedicated approach based on transportation problem for non-bipartite graphs which are also multigraphs, can be used to solve conflict problems with in-place architectures. For example, the data access matrix (Figure 4.26(a)) for shuffled decoding in the HSPA interleaver results in a multi-graph after applying *simple graph test*: as shown in Figure 4.26(b), as t_1 and t_3, t_2 and t_5, t_4 and t_5 have two parallel edges between them.

Theorem ***Bound for edge coloring multigraphs***

*Every multigraph with maximum degree P requires atmost (3/2)*P colors in any proper edge coloring [SHA49].*



(a)Data access matrix (b) Graph

Figure 4.26 Example resulting a multigraph

According to theorem presented above, a multigraph for a given memory mapping problem can be edge colored with atmost *(3/2)*P* colors which results in finding conflict free memory mapping using upto *(3/2)*P* banks. In order to edge color a multi-graph, we have proposed a simple polynomial time algorithm based on approach used for transportation problem using *(3/2)*P* banks which respects the bound introduced in [SHA49].

3.5.1. Modeling

In this section, we model our mapping problem as transportation problem (the modeling is same as shown in section 3.1). This transformation is carried out in two steps. In the first step, mapping problem is modeled as bipartite graph and different proves are provided in order to explain that it is always possible to divide this bipartite graph into different subgraphs of equal sizes (see [SAN11a]). Afterwards, this bipartite graph is transformed into transportation matrix. A transportation problem algorithm is applied using this matrix to find memory mapping.

The first step is to construct a bipartite graph $G_t = (T \cup L, E)$ in which vertex set T represents all the time instances and vertex set L represents all the data elements used in the computation. An edge $e = (t, l) \varepsilon E$ is incident to the data element vertex l and to the time instance vertex t if d needs to be processed at t (i.e. data l will be read and next written at time t). This bipartite graph has the same two properties described in section 3.1.

The bipartite graph for the data access matrix of Figure 4.26(a) is shown in Figure 4.27(a).

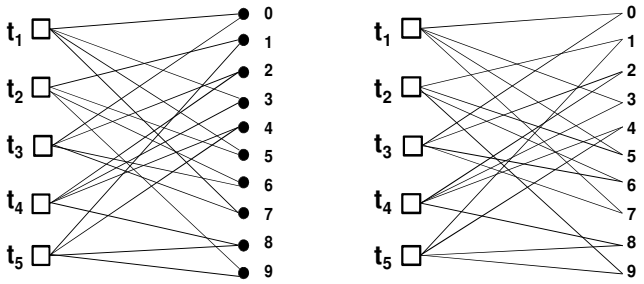


Figure 4.27 Bipartite Graph for example of Figure 4.26

In order to clearly explain how transportation problem is able to find conflict free memory mapping we introduce some definitions, theorems and corollaries.

Definition ***2-matching and 2-Factor of Graph***

2-matching H of a graph G = (T ∪ L, E) is a subset of E such that every node of G is incident with at most two edges of H. The 2-matching H of G is called 2-factor if every node of G is incident with exactly two edges of H [HAR06].

The proposed approach is based on finding semi 2-factor (defined below) to find conflict free memory mapping so we have introduced the following two theorems [GRO03] which define the necessary and sufficient condition for the graph to contain 2-factor.

Theorem4.1

Every 2k-regular graph contains a 2-factor, where k is integer.

Theorem4.2

Every 2-edge-connected (2k + 1)-regular graph contains a 2-factor.

These theorems result into the following two corollaries [GRO03]:

Corollary4.1

Every 2k-regular graph contains k 2-factors.

Corollary4.2

Every (2k + 1)-regular graph contains k 2-factors and one 1-factor.

Definition ***Semi 2-factor in Bipartite Graph***

A semi 2-factor in bipartite graph G is defined as a 2-regular sub-graph in G with 2Y vertices where every node is incident with exactly two edges and where Y = Min (|T|, |L|).

Corollary4.3

Every Bipartite Graph with $f_t = 2k$ or $f_t = 2k + 1$, where k is an integer, contains k disjoint semi 2-factors.

Proof: we first join the two edges connected with each data node and then remove all the data nodes to form regular graph $G_1 = (T, E_1)$ as shown in Figure 4.27(b). In this graph, $|E_1| = |L|$ i.e., each edge in G_1 corresponds to two edges or a data node in G_t. Since G_1 is regular, *2-factor* always exists in G_1 which implies that semi 2-factor of 2Y nodes where $|T| = Y$ always exists in G_t. Every 2-factor is a collection of cycles that spans all nodes of the graph going from 1 cycle with 2Y nodes up to Y/2 cycles of 4 vertices.

Additionally, each cycle c_i in G_1 can be even or odd which means c_i contains even or odd number of edges or time nodes.

Edges of every even cycle can be assigned with two colors which implies that edges in c_i and every 2-factor in G_1 can be colored with two colors [SOI08]. This results in the following lemma.

Lemma 4.1

All the data nodes in semi 2-factors of bipartite graph with even cycles can be assigned with two memory banks.

Moreover, edges of every odd cycle can be assigned with three colors which implies that edges in c_i and every 2-factor in G_1 can be colored with three colors [SOI08]. This results in the following lemma.

Lemma 4.2

All the data nodes in semi 2-factors of bipartite graph with odd cycle can be assigned with three memory banks.

As it is unknown that the cycle is odd or even therefore we consider the worst case that each cycle will be odd for every semi 2-factor which means that we will need three memory banks for each semi 2-factor. So k semi 2-factor will result in $3k$ banks i-e $(3/2)*P$ banks ($k = P/2$).

3.5.2. Transformation of bipartite graph into Transportation Matrix

Our graph is modeled as transportation matrix based on the properties discussed in the previous section. For this purpose, the bipartite graph is divided into *k semi 2-factors* and to give colors to the edges of each semi 2-factor. To find *semi 2-factor*, we transform our mapping problem as transportation problem by considering all the data nodes as producers and all the time nodes as consumers. The route l_{ij} exists between data node d_i and time node t_j if data d_i is accessed at t_j. One additional constraint must be considered while modeling our problem as transportation problem: the capacity of each route is fixed in our mapping problem. The reason is that each route represents a connection between processors and memory banks whose size is always fixed. In our case, the capacity x_{ij} of l_{ij} is kept one since only one data can be accessed at a given time instant t_j for this route.

In order to find *semi 2-factor,* we consider : (1) the demand of each consumer is kept to two and (2) each producer either provides two items (i.e. each data is accessed two times) or is not included in the current *semi 2-factor* (i.e. each producer must work at its full capacity). The cost o_{ij} of l_{ij} is kept one since the cost is not taken into account in the current work. It will only be used when we will consider the constraint of the network architecture. The matrix model for the bipartite graph of Figure 4.27(a) is shown in Figure 4.31(a). In this matrix, if the route l_{ij} does not exist between producer i and consumer j, then the corresponding cell M_{ij} is

kept empty. After construction of transportation matrix, any algorithm to solve transportation problem can be used to find semi-2 factor.

3.5.3. Algorithm to find semi 2-factors in Turbo Bipartite Graph

In this section, algorithm to solve memory mapping problem is presented. This algorithm is same as [SAN11b] with some modification. [SAN11b] is only able to tackle conflict problem for turbo codes (only non-shuffled) whereas our modified version of [SAN11b] is able to tackle all memory conflict problems. It traverses the transportation matrix to construct semi 2-factors. Flow diagram of the partitioning algorithm is shown in Figure 4.28.

The algorithm starts by first calculating the number of *semi 2-factors* i-e k by using the degree of each time node f_t of bipartite graph as explained in corollary 4.3. After that, it starts constructing the cycle (path) c_1 of current *semi 2-factor* sf_{cur} by choosing a first route l_{i1} connected with consumer t_1 (see Figure 4.31(a)). The selection of the route l_{i1} decreases the demand of t_1 and the supply of d_i to one. Simultaneously, the selected route is assigned with bank b_a where index a represent the next un-assigned bank ($a = 0$ for c_1). Algorithm then selects the route connected with t_1 choosing any route l_{k1} and assigns it with bank next bank b_{a+1}.

The selection of l_{k1} completes the demand of t_1, so all the producers connected with t_1 are completely removed from sf_{cur} because now they are unable to provide two items in sf_{cur} or they cannot work at their full capacity. The other route l_{km} connected with d_k is assigned the same bank b_{a+1} to reach to the consumer t_m. This completes the supply of d_k. Algorithm repeats the same process by selecting the route, decreasing the supply of producer and demand of consumer and alternately assigns banks to the route until c_1 is completed i.e., no producer with supply of one and no consumer with demand of one remains in the transportation matrix.

The final step is to check for the odd cycle for which a third bank b_{a+2} can be assigned. The odd cycle can be detected if the starting and ending route or routes are assigned with alternate banks which results in a conflict as two different banks cannot be assigned to same data. In this case the third bank b_{a+2} is assign to the starting and ending route of the cycle.

Furthermore, the algorithm tests whether all the consumers fulfill their demands. If not, the algorithm starts constructing another cycle c_2. For this, our algorithm selects consumer whose demand is still unfulfilled and which has at least one deleted route. Using this deleted route, the algorithm selects the route and assigns a bank b_a to this route. After the assignment

of b_a, the algorithm repeats the same process used for the construction of c_1 to complete c_2. When the algorithm finds that demands of all the consumers are fulfilled then it declares that sf_{cur} is constructed. In that case, the algorithm tests whether *k semi 2-factors* are constructed. If not, the algorithm removes sf_{cur} from transportation matrix, initializes all consumers with demand of 2 and starts constructing sf_{next} from remaining matrix using the process described above until *k semi 2-factors* are constructed. Partitioning algorithm is explained through a pedagogical example in the next section.

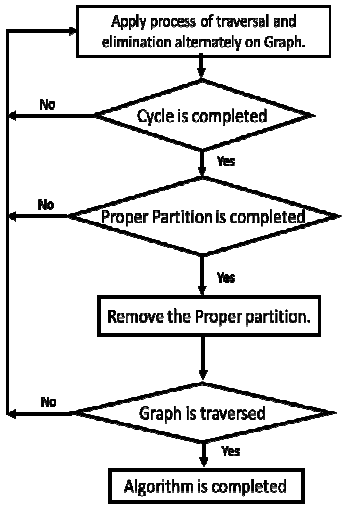


Figure 4.28 Partitioning algorithm

In [SAN11a], the author proposed an approach for finding semi 2-factor but their algorithm is not able to find cycles as explained with the help of an example shown in Figure 4.29. In this example, the transportation matrix is constructed first as shown in Figure 4.29(b). Then, the algorithm is applied on this matrix as explained for constructing cycle c_1. The algorithm is based on depth first algorithm without any recursion in which the cycle is started at a node and the next node is selected randomly. But it can be seen that there is a problem at t_6 in the cycle. Two edges are deleted at t_6 instead of deleting one edge which is wrong solution for finding cycles. So, we have no solution in this case using approach [SAN11a].

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
PE_1	0	1	2	3	1	0	4	7
PE_2	4	5	6	7	3	2	5	8
PE_3	8	9	10	11	11	6	10	9

(a)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	
0	$l_{01}(1)$					$l_{06}(1)$			2
1		$l_{12}(1)$			$l_{15}(1)$				2
2			$l_{23}(1)$			$l_{26}(1)$			2
3				$l_{34}(1)$	$l_{35}(1)$				2
4	$l_{41}(1)$						$l_{47}(1)$		2
5		$l_{52}(1)$					$l_{57}(1)$		2
6			$l_{63}(1)$			$l_{66}(1)$			2
7				$l_{74}(1)$				$l_{78}(1)$	2
8	$l_{81}(1)$							$l_{88}(1)$	2
9		$l_{92}(1)$						$l_{98}(1)$	2
10			$l_{103}(1)$				$l_{107}(1)$		2
11				$l_{114}(1)$	$l_{115}(1)$				2
	2	2	2	2	2	2	2	2	

(b)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	
0	$l_{01}(b_0)$					l_{06}			2
~~1~~		~~l_{12}~~			~~l_{15}~~				~~2~~
2			l_{23}			l_{26}			2
~~3~~				~~l_{34}~~	~~l_{35}~~				~~2~~
4	$l_{41}(b_1)$						$l_{47}(b_1)$		2
5		$l_{52}(b_0)$					$l_{57}(b_0)$		2
6			l_{63}			l_{66}			2
7				$l_{74}(b_0)$				$l_{78}(b_0)$	2
~~8~~	~~l_{81}~~							~~l_{88}~~	~~2~~
9		$l_{92}(b_1)$						$l_{98}(b_1)$	2
~~10~~			~~l_{103}~~				~~l_{107}~~		~~2~~
11				$l_{114}(b_0)$	$l_{115}(b_1)$				2
	2	2	2	2	2	2	2	2	

(c)

Figure 4.29 Cycle construction problem for [SAN11a] coloring algorithm

In this thesis, we have proposed another memory mapping approach based on transportation problem using *breadth first algorithm* with recursion. In the proposed approach, the cycle is started from a node and all the nodes are recorded at that node. Then, one node is selected and the cycle is completed using the selected node. However, if the cycle is not completed or the cycle results in the deletion of additional rows then other recorded nodes can be explored one by one in the tree *(breadth first algorithm)* to construct a cycle until the cycle is completed. As in Figure 4.29 the cycle is not completed due to a problem at t_5 so we have selected another node at t_1 and assigned b_1 to data 8 instead of data 4. In this case, the cycle is completed without any problem as shown in Figure 4.30.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	
1	b_0					b_0			2
~~1~~		~~1(1)~~			~~1(1)~~				~~2~~
~~2~~			~~1(1)~~			~~1(1)~~			~~2~~
3				1(1)	1(1)				2
4	~~1(1)~~						~~1(1)~~		~~2~~
5		b_1					b_1		2
6			b_1			b_1			2
~~7~~				~~1(1)~~				~~1(1)~~	~~2~~
8	b_1							b_1	2
9		b_0						b_0	2
11			b_0				b_0		2
11				1(1)	1(1)				2
	2	2	2	2	2	2	2	2	

Figure 4.30 Resulting cycle obtained with proposed cycle construction approach

Complexity of the algorithm

Algorithm needs to traverse f_t edges at each time instance to select two accesses that can be included in sf_{cur}. However we need to use *breadth first algorithm* to explore all the option, so its complexity is O (f_t + |T|) where |T| is the number of time nodes. To construct a partition sf_{cur}, algorithm needs to select a couple of accesses for each time nodes So, the number edges to be traversed for one partition is in the worst case (f_t + |T|) $*$ |T|. Since there are $k = f_t/2$ semi-2 factors (see definition), in order to construct all the partitions, the overall complexity of the partitioning algorithm is O ($f_t/2 * |T| * (f_t + |T|)$.

3.5.4. Pedagogical Example

We present an example based on data access matrix depicted in Figure 4.26. The first step is to constructs the bipartite graph which is shown in Figure 4.27(a). This semi regular bipartite graph has time nodes with degree $ft = 4$. There are two semi 2-factors using corollary 4.3. The second step first transforms the bipartite graph into matrix model of the transportation problem which is depicted in Figure 4.31(a).

The algorithm starts constructing the cycle c_1 from the first route l_{01} (data d_0 connected with time node t_1) and assigns the memory bank b_0. Since one route is occupied, the algorithm

reduces the supply and demand to 1 in the matrix as shown in Figure 4.31(b). The algorithm then fulfills the demand of t_1 by choosing a data elements in t_1. The algorithm selects a route l_{31} (data d_3 connected with time node t_1) and assigns another memory bank b_1. Now the demand of t_1 is completely fulfilled and the remaining producers connected with t_1 (d_5 and d_7 in this case) are completely removed because these producers are unable to work at their full capacity (see Figure 4.31(c)).. The second route connected with producer d_3 (l_{34}), is also assigned the same bank b_1. The algorithm fulfils the supply and demand of producers and consumers respectively in the same manner until the cycle c_1 is completed (cycle is completed when we will reach at l_{03}) i.e the algorithm do not contain any producer with supply of 1 and any consumer with demand of 1 in the transportation matrix. This process is presented in Figure 4.31(c). However, as shown in this figure that producer d_0 is assigned with different memory banks b_0 and b_1 which is in-valid for mapping as one data could not be mapped in two banks. Therefore, an additional bank is needed for d_0 so we will assign b_2 for d_0 both at *t1* and *t3* as shown in Figure 4.31(d).

	t_1	t_2	t_3	t_4	t_5	
0	l_{01}		l_{03}			2
1		l_{12}			l_{15}	2
2			l_{23}	l_{24}		2
3	l_{31}			l_{34}		2
4				l_{44}	l_{45}	2
5	l_{51}	l_{52}				2
6		l_{62}	l_{63}			2
7	l_{71}		l_{73}			2
8				l_{84}	l_{85}	2
9		l_{92}			l_{95}	2
	2	2	2	2	2	

(a)

	t_1	t_2	t_3	t_4	t_5	
0	l_{01}(b0)		l_{03}			1
1		l_{12}			l_{15}	2
2			l_{23}	l_{24}		2
3	l_{31}			l_{34}		2
4				l_{44}	l_{45}	2
5	l_{51}	l_{52}				2
6		l_{62}	l_{63}			2
7	l_{71}		l_{73}			2
8				l_{84}	l_{85}	2
9		l_{92}			l_{95}	2
	1	2	2	2	2	

(b)

	t_1	t_2	t_3	t_4	t_5	
0	l_{01}(b0)		l_{03}(b1)			X
1		l_{12}			l_{15}	2
2			l_{23}(b0)	l_{24}(b0)		X
3	l_{31}(b1)			l_{34}(b1)		X
4				l_{44}	l_{45}	2
5	l_{51}	l_{52}				2
6		l_{62}	l_{63}			2
7	l_{71}		l_{73}			2
8				l_{84}	l_{85}	2
9		l_{92}			l_{95}	2
	X	2	X	x	2	

(c)

	t_1	t_2	t_3	t_4	t_5	
0	l_{01}(b2)		l_{03}(b2)			X
1		l_{12}			l_{15}	2
2			l_{23}(b0)	l_{24}(b0)		X
3	l_{31}(b1)			l_{34}(b1)		X
4				l_{44}	l_{45}	2
5	l_{51}	l_{52}				2
6		l_{62}	l_{63}			2
7	l_{71}		l_{73}			2
8				l_{84}	l_{85}	2
9		l_{92}			l_{95}	2
	X	2	X	x	2	

(d)

Figure 4.31 Approach based on transportation problem (part-1)

The algorithm continues to find cycle c_2 by using the same approach used in construction of c_1 until the cycle c_2 is completed as shown with gray highlighted cells in Figure 4.32(a). We need only one additional bank for all cycles in a semi 2-factor as

additional bank used in c_1 can be reused in case of an odd cycle of c_2 (not in case of this example).

	t_1	t_2	t_3	t_4	t_5	
0	$l_{01}(b2)$		$l_{03}(b2)$			X
1		$l_{12}(b0)$			$l_{15}(b0)$	X
2			$l_{23}(b0)$	$l_{24}(b0)$		X
3	$l_{31}(b1)$			$l_{34}(b1)$		X
4				~~l_{44}~~	~~l_{45}~~	~~2~~
5	~~l_{51}~~	~~l_{52}~~				~~2~~
6		~~l_{62}~~	~~l_{63}~~			~~2~~
7	~~l_{71}~~		~~l_{73}~~			~~2~~
8				~~l_{84}~~	~~l_{85}~~	~~2~~
9		$l_{92}(b1)$			$l_{95}(b1)$	X
	X	X	X	x	X	

(a)

	t_1	t_2	t_3	t_4	t_5	
0						
1						
2						
3						
4				$l_{44}(b3)$	$l_{45}(b3)$	X
5	$l_{51}(b5)$	$l_{52}(b5)$				X
6		$l_{62}(b3)$	$l_{63}(b3)$			X
7	$l_{71}(b4)$		$l_{73}(b4)$			X
8				$l_{84}(b4)$	$l_{85}(b4)$	X
9						
	X	X	X	X	X	

(b)

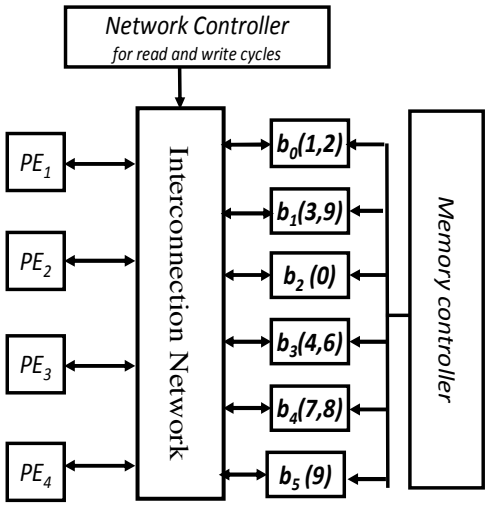
Figure 4.32 Approach based on transportation problem (part-2)

The demands of all the consumers are fulfilled so the first semi 2-factor is constructed. The algorithm removes the first semi 2-factor from the transportation matrix. It initializes all consumers with demand of 2 and starts constructing the second semi 2-factor from remaining matrix using the process described for the first semi 2-factor as shown in Figure 4.32(b) in which there are shown two cycles c_1 and *c2* (in Figure 4.32(b) c_1 is shown in plane and c_2 is shown in gray cells). c_2 is also odd cycle so we also need an extra bank to map it.

The final mapping is shown in Figure 4.33(a) in which six *((3/2)*P)* banks are used to find conflict free memory mapping. The Figure 4.33(b) gives the final architecture along with the memory mapping for the considered example.

	t_1	t_2	t_3	t_4	t_5	
0	$l_{01}(b2)$		$l_{03}(b2)$			X
1		$l_{12}(b0)$			$l_{15}(b0)$	X
2			$l_{23}(b0)$	$l_{24}(b0)$		X
3	$l_{31}(b1)$			$l_{34}(b1)$		X
4				$l_{44}(b3)$	$l_{45}(b3)$	X
5	$l_{51}(b5)$	$l_{52}(b5)$				X
6		$l_{62}(b3)$	$l_{63}(b3)$			X
7	$l_{71}(b4)$		$l_{73}(b4)$			X
8				$l_{84}(b4)$	$l_{85}(b4)$	X
9		$l_{92}(b1)$			$l_{95}(b1)$	X
	X	X	X	X	X	

(a) Final mapping matrix



(b) Final mapping architecture

Figure 4.33 Mapping based on transportation problem

4. Experiments and results

Different experiments have been performed to validate the theoretical work presented in this chapter. Again, all the results in this are given in NAND-gate equivalent area using 90nm technology from STMicroelectronics. These estimations are based on synthesized and pre-characterized components (Registers, multiplexers, …). The number of the different components is provided by the mapping tool and as a result the estimations for the architecture are generated.

We have performed experiments for each cases of the proposed approach considering three test cases: shuffled turbo decoders for *LTE,* non-binary *LDPC* and shuffled turbo decoders for *HSPA*.

4.1. Case study-1: Shuffled turbo decoders for *LTE*

In the first part of chapter-3, we have presented a case study for 3GPP-LTE standard turbo decoder for nine different configurations as shown in Table.4.1. In-place memory mapping architecture was used to solve configuration 1 to 5 (non-shuffled architecture) and *MRMW* memory mapping architecture was used to solve configuration 6 to 7 (shuffled architecture) [SAC12] with *L = 1024 and P = 16 and 32*. We have applied our proposed approach to solve configuration 6 to 7 (shuffled architecture) to use in-place memory mapping architecture.

Table 4.1 Different configuration to explore the design space for turbo decoding

	Mode	Scheduling	Radix	Internal Memory
Config. 1	Non-Shuffled	Butterfly	2	YES
Config. 2	Non-Shuffled	Butterfly	4	YES
Config. 3	Non-Shuffled	Butterfly	16	YES
Config. 4	Non-Shuffled	Butterfly	2	No
Config. 5	Non-Shuffled	Butterfly	4	No
Config. 6	Shuffled	Replica	2	No
Config. 7	Shuffled	Replica	2	YES
Config. 8	Shuffled	Replica	4	No
Config. 9	Shuffled	Replica	4	YES

Bipartite test is applied to test the bipartiteness of these configurations as shown in Figure 4.5. Results showed that these configurations are of bipartite nature and now we can apply bipartite edge coloring algorithm using in-place memory mapping architecture [SAN13]. In [SAC12], *MRMW* mapping architecture was used to solve configuration 6-9 but thanks to our proposed design flow we can solve the conflict problems for these configurations using in-

place architecture by applying the proposed bipartite test. The area comparison is presented in Figure 4.34 for configuration 6 to 9. The cost of the architecture (excluding processors) can be reduced up to 25% of the total area by using our proposed approach. The reduction in case of $P = 32$ is almost the same.

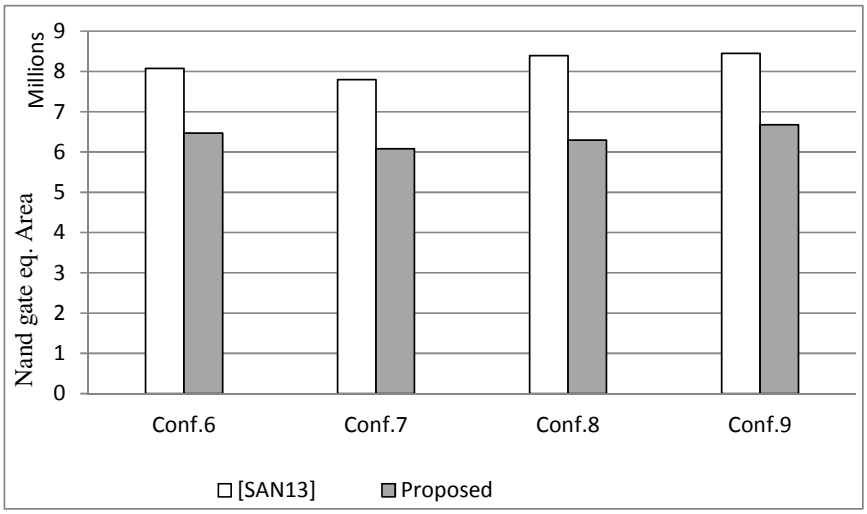


Figure 4.34 Area comparison for shuffled turbo decoders with $P=16$

4.2. Case study-2: Non-Binary *LDPC* codes

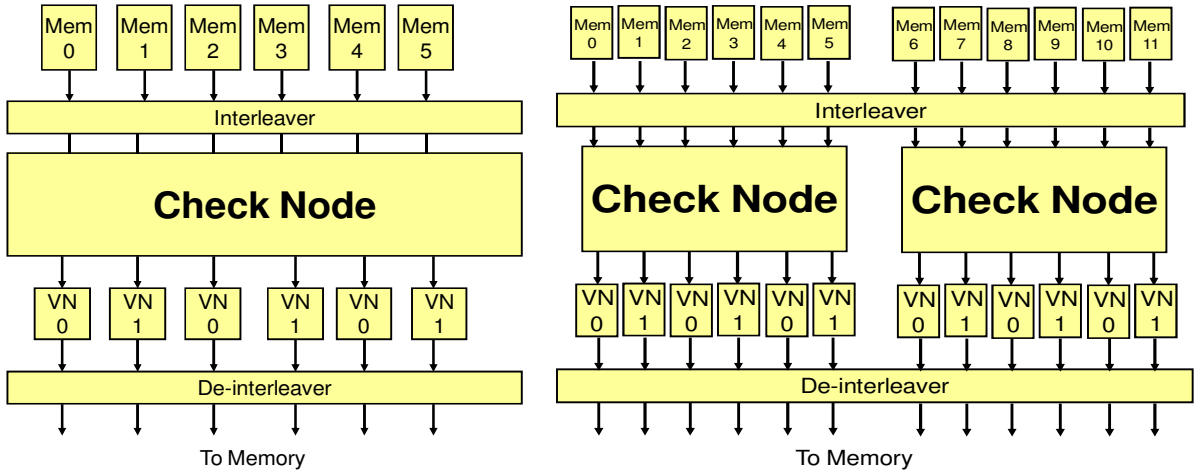
An extension of binary *LDPC* codes has been developed to further reduce the gap of performance with Shannon limit. This new class of codes is known as non-binary *LDPC* (NB-LDPC) codes [DAV]. These codes improve the performance of small and moderate codeword lengths. However, increase in decoder complexity for NB-LDPC motivates to develop decoding algorithms that are easily implementable. Also, unlike structured codes, routing of the edges of tanner graph is not regular and even implementing NB-LDPC on serial architecture suffers from memory conflict problem.

The DAVINCI project [DAV], funded by the European Commission under the seventh framework (FP7) of collaborative research, designed the novel NB-LDPC codes and related link level technologies. The purpose of this project was to construct codes that are suitable for implementation and outperform the state of the art techniques to design NB-LDPC codes.

Typical serial decoder architecture [DAV] for NB-LDPC codes developed in DAVINCI project is shown in Figure 4.35.a. This decoder is used to decode NB-LDPC codes with check node degree = d_c = 6 and variable node degree = d_v = 2. The decoder consists on one CN processor and six VN processors. The decoder is designed based on serial implementation to process one check node at each cycle. To achieve high memory bandwidth, main memory is divided into d_c number of memory banks to simultaneously receive d_c messages from

memory. The interleaver and the deinterleaver are designed to transfer data between CN processors, VN processors and memory banks.

For partially parallel architecture, two check nodes are processed in parallel at the same time as shown in Figure 4.35.b. The main memory is divided into $2*d_c$ number of memory banks to concurrently fetch $2*d_c$ messages from memory.



(a) Serial Architecture (b) Partially Parallel Architecture

Figure 4.35 Architecture for NB-LDPC

The problem is to allocate messages into memory banks in such a manner that at each cycle CN processor can fetch d_c number of messages from d_c number of memory banks concurrently without any conflict.

The first step is to prepare data access matrix for the architectures. Data access matrix for partially parallel architecture is shown in Figure 4.36 where 6 data elements are needed to be accessed in parallel for one check node (12 data are needed for two check nodes). To find conflict free memory mapping, all data should be stored in memory bank in such a manner that there is no conflict in accessing them in each cycle.

2	10	62	23	59	29	33	26	25	16	17	63	36	21	15	11	18	9	38	34	40	31	3	12	41	32	39	13	30	22	43	60
3	11	63	24	60	30	34	27	26	17	18	64	37	22	16	12	19	10	39	35	41	32	4	13	42	33	40	14	31	23	44	61
149	139	159	67	156	97	142	130	115	83	155	65	78	132	181	116	96	174	109	98	106	125	183	91	117	169	123	74	190	189	162	110
122	123	84	114	133	110	70	119	181	127	109	10 6	184	98	68	96	156	83	85	190	169	78	91	115	81	67	134	108	117	136	167	189
90	86	126	178	73	191	172	80	153	128	152	176	157	171	107	140	175	99	170	82	161	188	138	173	168	144	95	145	66	75	105	147
157	179	182	158	124	141	88	148	66	143	75	131	103	126	144	178	105	129	145	138	77	135	128	161	163	118	180	188	152	113	82	99
48	19	53	42	51	47	46	7	58	28	5	6	1	50	44	35	61	20	56	37	54	14	45	55	8	57	27	49	52	64	4	24
49	20	54	43	52	48	47	8	59	29	6	7	2	51	45	36	62	21	57	38	55	15	46	56	9	58	28	50	53	1	5	25
146	104	112	79	113	85	81	84	184	70	167	101	136	108	87	102	119	68	164	154	121	133	71	100	114	120	127	122	72	135	185	94
100	94	90	154	183	72	164	146	142	159	130	155	139	71	112	174	149	97	116	121	132	102	101	162	74	171	87	79	125	104	65	185
88	160	137	129	148	177	166	103	89	111	118	93	92	141	131	77	180	134	69	165	179	186	192	143	187	151	163	158	182	76	124	150
153	177	172	151	89	147	107	168	95	170	86	173	111	175	186	15	69	187	191	80	73	92	165	76	137	93	160	176	140	192	166	120
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}	t_{31}	t_{32}

Figure 4.36 Data Access Matrix for L = 192 and d_c = 6

4.2.1. Vizing theorem for non-binary LDPC codes

To solve memory conflict problem for NB-LDPC codes, MRMW architecture has been used in [SAN12]. However, the approach we propose in this chapter can solve memory mapping problem for non-binary LDPC using in-place mapping architecture.

***Discussion**:*

At-most P+1 banks are needed for memory conflict problem in non-binary LDPC codes by using Vizing theorem.

As already shown, Vizing theorem needs $P+1$ banks for memory conflict problems only with simple graphs. So a memory mapping problem from non-binary LDPC codes can be solved with $P+1$ banks using Vizing theorem if non-binary LDPC mapping problem results into a simple graph. As simple graph have no parallel edges and there will be no parallel edges in a graph if any two time nodes have only one data common between them. As it can be seen in Figure 4.21a(and simpler example in Figure 4.7) that the graph is a simple graph as the data access matrix Figure 4.20 has only one data element common between two time nodes. Now we have to prove that only one data element is common between two time nodes in non-binary LDPC codes.

Girth is an important parameter for LDPC codes, which is strongly related to code performance. Girth can be defined as the size of the smallest cycle of the bipartite graph. Girth $g <= 4$ are normally considered as short cycles in LDPC codes. Short cycles are avoided in LDPC codes as it degrades the performance of the decoder [XIO05]. If $g <= 4$ then more than one data elements is common between two time nodes (check nodes) which will result in parallel edges. So, for better performance LDPC codes are designed with $g > 4$. Therefore, the data access matrix will always have only one data element common between two time nodes as it can be seen in the example shown in Figure 4.20 with $g > 4$. In this example, the data access matrix have only one data element common between two time nodes. This property will make the resultant graph into a simple graph because only one edge will be possible between two nodes (due to one same data). Therefore, Vizing theorem can be applied to solve the memory conflict problem for non-binary LDPC codes.

4.2.2. Results

We have performed experiments for NB-LDPC with different block lengths and parallelisms. In-place memory mapping solutions in the state of art approaches like [CHA10a] are not able to find conflict free memory mapping for NB-LDPC conflict problem. Therefore

we compared our proposed work with [CHA10b] and [BRI13b]. The result for [SAN13] are not shown as they are same as [CHA13b] because mapping solution with [SAN13] for these experiments are also based on *MRMW* architecture so the cost of the architecture will be same for [SAN13] and [CHA10b]. The approach proposed in [BRI13b] is also to generate optimize hardware architecture so we have compared our results with [BRI13b].

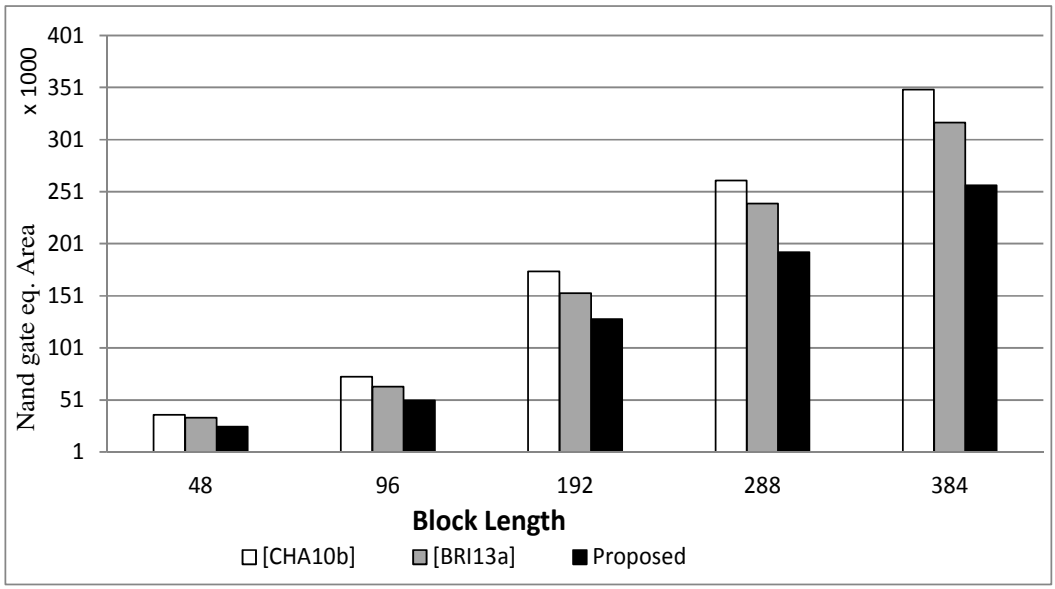


Figure 4.37 Comparison of NB_LDPC decoder areas obtained with state of art approaches and Vizing coloring for different block length ($P = 6$)

The results are shown in Figure 4.37 and Figure 4.38. The results for different block lengths with $P= 6$ are shown in Figure 4.37 and with $P= 12$ in Figure 4.38. The approach [BRI13a] optimizes the architecture up to 15% whereas our proposed approach can optimize the architecture up to 40% as compared to approaches without optimization [CHA10b].

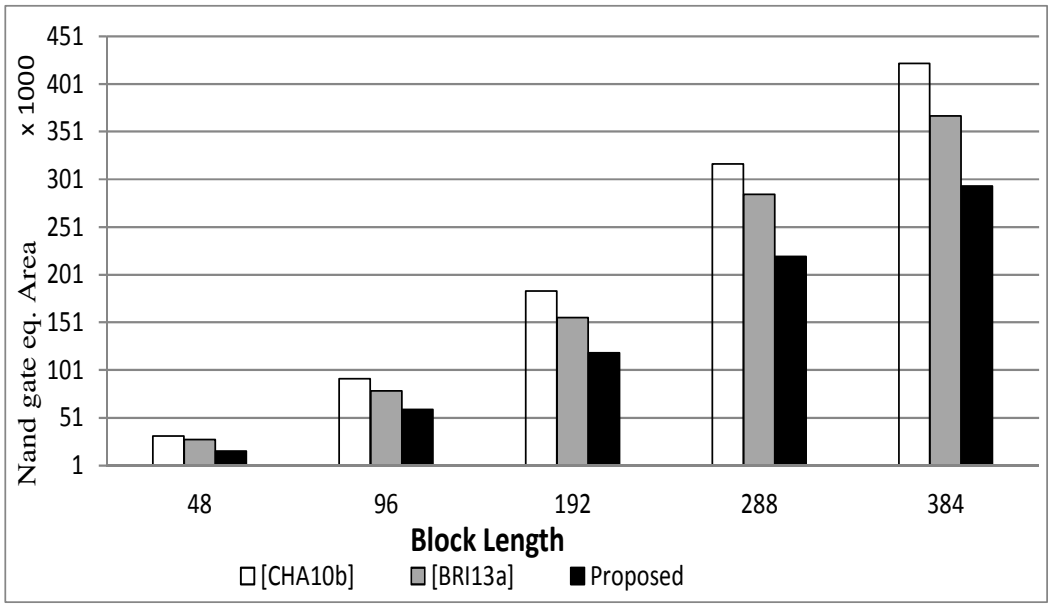


Figure 4.38 Comparison of NB_LDPC decoder areas obtained with state of art approaches and Vizing coloring for different block length (P = 12)

4.3. Case study-3: Shuffled turbo decoding for HSPA

We have discussed the shuffled decoding for turbo decoding in chapter-2 in details. We have also explored the memory conflict problem for *LTE* in section 4.1 of this chapter. The turbo decoder architecture given in chapter-2 is same for *LTE* and *HSPA* interleaver. However, the memory conflict problem is different in both the cases.

We have applied our proposed approach for shuffled turbo decoding in *HSPA* interleaver to solve the memory conflict problem. According to the design flow shown in section-3, first we have applied the bipartite test. The resultant graph in this case was not a bipartite graph. Then, we have applied the simple graph test and we have observed that Vizing theorem cannot be applied as the resultant graph is a multigraph. Therefore, the approach based on transportation problem is used to solve the memory conflict problem for this test case as the resultant graph.

In Figure 4.39, we show first results based on our approach. We have considered the worst case scenario for these results as our transportation problem based memory mapping approach uses 3/2*P memory banks each time. In fact, these are the worst case results as less than 3/2*P could also be used for a given test case, but the current software version of the proposed design flow is not fully optimized.

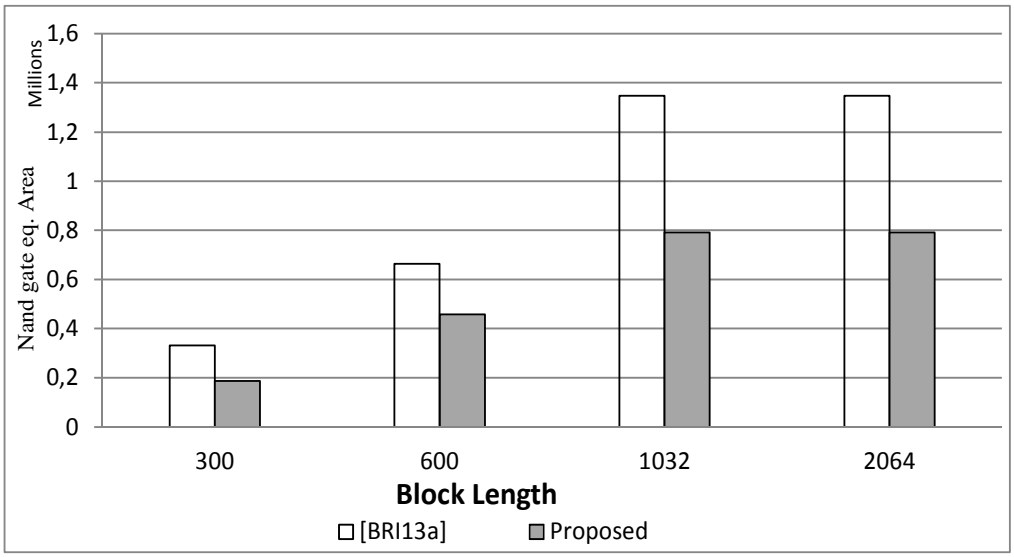


Figure 4.39 Area comparison for different block length for HSPA P = 16 (best results)

However, even in worst cases, in place memory mapping can reduce the cost up to 37% as compared with [BRI13a] as shown in Figure 4.39. The proposed approach needs up to six banks instead of four memory banks (in case of [BRI13a]) to find conflict free memory mapping. However, due to the in-place architecture the final cost is still reduced. As [BRI13a] is an optimization of the approach [CHA10b] so we have compared our results with [BRI13a] only for the block lengths in which the current software of [BRI13a] provides optimized results than [CHA10b].

5. Conclusion

In this chapter, we have shown that all conflict problems with two accesses can be solved using in-place architecture. We have proposed algorithms to find conflict free memory mapping for two accesses conflict problem by using in-place architecture as state of art approaches needs *MRMW* architecture for some of these memory mapping problems. Hence, the total area of the decoder can be reduced by using in-place architecture for memory mapping problems. We have proposed *Vizing theorem* for solving memory conflict problem using in-place architecture. Moreover, we have introduced a dedicated approach based on transportation problem to solve memory conflict problems using in-place architecture. These proposed algorithms are able to find conflict free memory mapping in polynomial time.

In future, we can further optimize the decoder architecture by merging our two proposed optimization concepts: the network customization and in-place memory architecture. We could further optimize the solutions by using the customized network approach using in-place

memory mapping architectures. The second perspective is to extend the proposed in-place approach for other test cases with more than two data accesses.

In design time approaches, ROM blocks are used to control interconnection network and generating addresses for different memory banks which may be sufficient to design parallel architecture that supports single code-word or applications. However, to design hardware architecture that supports complete standard and/or different applications, ROM based approach results in huge hardware cost and area. To reduce hardware cost, optimizations are required to use as less ROMs as possible to support different applications. For this purpose, an approach based on on-chip memory mapping mechanism is proposed in the next chapter.

Chapter 5

ON-CHIP IMPLEMENTATION OF MEMORY MAPPING ALGORITHM TO SUPPORT FLEXIBLE DECODER ARCHITECTURE

Table of Contents

In this chapter, we propose on-chip memory mapping approach in order to reduce the requirement of the multiple ROM blocks. Design time approaches need ROM blocks to store the control configurations for memory and network. However, multiple memory elements are required to support different block lengths/different standards which results in high complexity(area). In order to overcome this problem, we have proposed to embed memory mapping approaches on-chip to solve memory conflict problem in parallel hardware decoders. Dedicated architecture composed of an embedded processor and RAM to store command words are proposed. We propose to embed on-chip the polynomial time memory mapping approach and a routing algorithm based on Benes network to solve memory conflict problem in parallel hardware decoders. Different experiments are performed by using memory mapping approaches executed on several embedded processors and results are presented.

1. Introduction

Design time approaches find memory mappings that provide conflict free concurrent access to all the memory banks. In these approaches, ROM blocks are needed to store the network, memory and other control configurations. However, multiple ROM blocks are needed to support different block lengths within a standard or multiple standards which results in high hardware cost. As shown in Figure 5.1, multiple ROM blocks are needed to store address generation logic and network control logic to support multiple block lengths and/or multiple applications. This results in huge hardware cost that is utilized in storing addressing, network and other control logic to design flexible decoder architecture. In order to reduce hardware cost, optimization is required to store addressing and control logic for multiple block lengths or multiple applications. Unfortunately, state of the art memory mapping approaches are unable to optimize memory necessary to store control information for multiple blocks lengths or applications.

In order to overcome the hardware overhead problem, we propose a solution to run mapping approaches on chip in order to calculate new mapping information on the fly as soon as new block length needs to be decoded and to update these new generated control information in memory. This work has been published in [REH12] [REH14b].

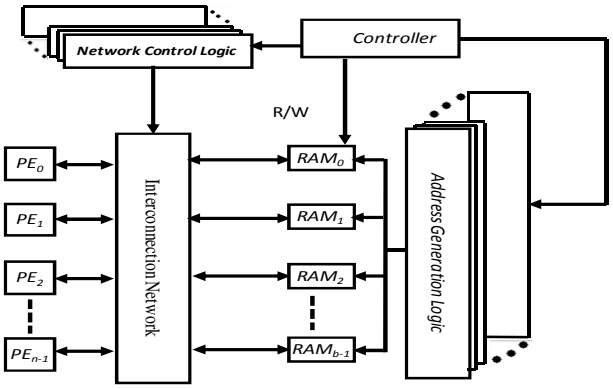


Figure 5.1 Parallel decoder architecture

2. On-chip implementation of memory mapping algorithms

The proposed dedicated hardware architecture is shown in Figure 5.2 for embedding memory mapping algorithms on chip. Control unit includes a dedicated processing element (General Purpose Processor GPP, Application Specific Instruction set Processor ASIP or Application Specific Integrated Circuit ASIC) to execute the mapping algorithm. The architecture is shown in Figure 5.2 in which multiple network and addressing ROMs are replaced by a two RAMs i.e. *Network RAM* and *addressing RAM*. Control Unit executes the

mapping algorithm and updates these RAMs when each time block length changes or the application changes.

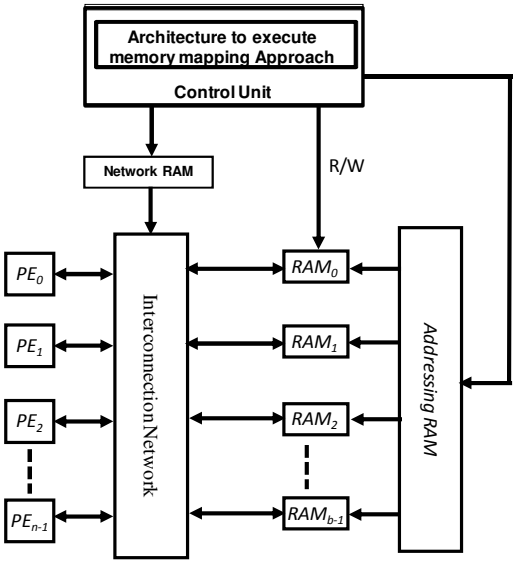


Figure 5.2 Parallel decoder architecture to embed memory mapping algorithms on chip

Sizes of *Network* and *addressing* RAMs depend on maximum block length and on the parallelism supported by decoding architecture. To determine size of different components of the architecture to support complete telecommunication standard, the following parameters are considered:

P = Total Number of processing elements

B = Total number of memory banks

T = Total number of access to the memory

M = Maximum number of data in each bank

Size of addressing RAM = $B * T * \lceil log_2(M) \rceil$, where size of each word is $B*\lceil log_2(M) \rceil$. bits. Similarly, for Benes network, the size of network RAM = $T * (B/2*((2*\log_2 B)-1))$. Also the size of bus from network RAM to network is $B/2*((2*\log_2 B)-1)$ bits and the size of each bus from addressing RAM to bank is $B*\lceil log_2(B) \rceil$ bits.

The proposed approach is based on the design flow introduced below.

2.1. Proposed Design flow

The proposed design flow is shown Figure 5.3. In the first step, the data access order is generated based on the input parameters like interleaving law, block length sizes, level of parallelism and scheduling. The second step is the execution of the memory mapping approach using the data access order from the previous step. As a result a conflict free

memory mapping is generated which contains the memory address information. The final step is to generate the routing information for the interconnection network. We will define each of these steps in details.

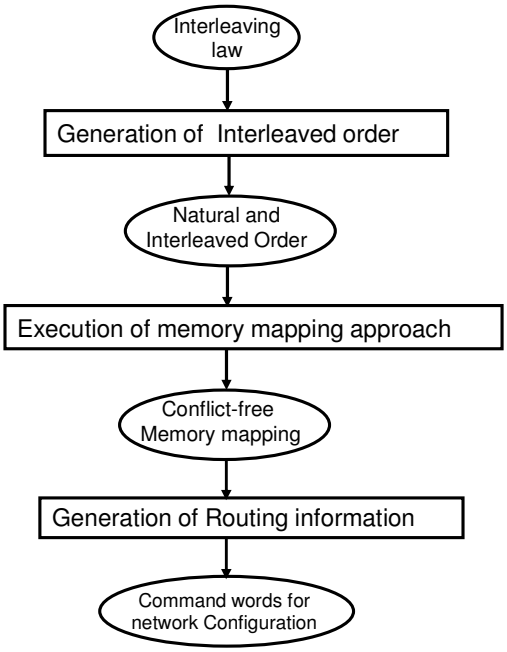


Figure 5.3 Embedded conflict free memory mapping flow

2.2. Generation of data access order

The first step of the design flow is to generate data access order based on the input parameter like interleaving law, block length size and parallelism. To illustrate, we have considered two interleavers: LTE and HSPA. Here we will describe in detail how to generate the interleaved order for both of these interleavers.

a. HSPA Interleaver

Series of specification have been released from time to time for *high speed packet access* (HSPA) after the release of initial draft by 3GPP-WCDMA. To obtain high throughput, it is necessary to perform turbo decoding on parallel architecture. However, the interleaver used in HSPA+ is not conflict free to support parallel implementation of turbo decoder. Also it is necessary to design interleaver architecture that support wide range of block sizes used in HSPA+ i.e. from 40 to 5114.

The interleaving algorithm for HSPA+ defined in [HSP04] is mentioned below:

L Number of bits input to Turbo code internal interleaver

R Number of rows of rectangular matrix described in standard

C Number of columns of rectangular matrix described in standard

p Prime number described in standard

v Primitive root describe in standard

- Determine *R* of the rectangular matrix, such that

$$R = \begin{Bmatrix} 5, if\ (40 \le K \le 159) \\ 10, if\ ((160 \le K \le 200) or (481 \le K \le 530)) \\ 20, if\ (K = \text{any other value}) \end{Bmatrix}$$

- Determine value of *p* and *C*, such that

if (481 ≤ L ≤ 530) then

p = 53 and C = p

else

Find *p* from Table 5. 1 such that

$L \le R*(p + 1)$,

and determine *C* of matrix such that,

$C = p - 1;$ *if* $(L \le R * (p-1))$

$C = p;$ *if* $(R*(p-1) < N \le R * p))$

$C = p + 1;$ *if* $(R*p < N)$

Table 5. 1. List of prime number *p* and associated primitive root *v*

p	*v*	*p*	*v*	*p*	*v*	*p*	*v*	*p*	*v*
7	9	47	5	101	2	157	5	223	3
11	2	53	2	103	5	163	2	227	2
13	2	59	2	107	2	167	5	229	6
17	3	61	2	109	6	173	2	233	3
19	2	67	2	113	3	179	2	239	7
23	5	71	7	127	3	181	2	241	7
29	2	73	5	131	2	191	19	251	6
31	3	79	3	137	3	193	5	257	3
37	2	83	2	139	2	197	2		
41	6	89	3	149	2	199	3		
43	3	97	5	151	6	211	2		

- Write the input bit sequence into the rectangular matrix row by row and if R*C > L, the dummy bits are padded to fill the matrix.
- Construct the base sequence *S(j)* for intra-row permutation as:

*S(j) = [v * S(j-1)] % p;* where *j = 1,2, ……., p-2*

- Determine the least prime integer sequence *q(i)* for *i = 1, 2, ……., R-1* , by assigning *q(0) = 1,* such that *gcd(q(i), p-1) = 1* and *q(i) > 6* and *q(i) > q(i-1).*
- Permute the sequence q(i) to construct the sequence *r(i)* such that

$r_{T(i)} = q(i)$ where *i = 0, 1, …., R-1* and *T(i)* is the inter-row permutation defined in the standard.

- Perform the intra row permutation $U_i(j)$, such that

for *i = 0, 1,….., R-1* and *j = 0,1, ……, p-2*;

If *(C = p)* then

$U_i(j) = S[(j*r(i)) \bmod (p-1)]$ and $U_i(p-1) = 0;$

If *(C = p+1)* then

$U_i(j) = S[(j*r(i)) \bmod (p-1)]$ and $U_i(p-1) = 0$ and $U_i(p) = p$

and *if (L = R*C)* then exchange $U_{R-1}(p)$ *with* $U_{R-1}(0)$

if *(C = p-1)* then

$U_i(j) = S[(j*r(i)) \bmod (p-1)] - 1$

- Perform the inter row permutation of the matrix based on the pattern *T(i)* where *T(i)* is the original row position of the *i-th* permuted row and defined in the standard.
- Read the bits column by column from the rectangular matrix by deleting the dummy bits padded to the input bits sequence.

Example for HSPA Interleaver

The algorithm can be explained best through a small example of $L = 44$. Different parameters obtained from the specifications explained previously are:

$$R = 5, C = 10, p = 11, v = 2$$

Next step is to put 44 data into matrix of order 5*10 (R*W) starting from row 0. Since there are 50 cells in the matrix, so the last 6 cells are filled with dummy bits represented by -1 in the last row as shown in Figure 5.4.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	-1	-1	-1	-1	-1	-1

Figure 5.4 Arrangement of $L = 44$ data into 5*10 matrix

Afterward, values for sequences *s, q, r, u* are calculated based on the rules defined in the standard. These values are:

S = 1 2 4 8 5 10 9 7 3 6 where number of values in S is $(p-1) = 10$

q = 1 7 11 13 17 where number of values in q is $R = 5$

r = 17 13 11 7 1 where number of values in q is $R = 5$

Value of U

0 6 4 1 2 9 3 5 8 7

0 7 8 5 3 9 2 1 4 6

0 1 3 7 4 9 8 6 2 5

0 6 4 1 2 9 3 5 8 7

0 1 3 7 4 9 8 6 2 5

where number of values in *U* is *R*C* = 5*10 = 50

The values in *U* are used to perform intra-row permutation. First row of *U* values are used to permute values in first row of matrix. For the values calculated for this example, first value remains at the first place, second value is permuted to sixth value, third value is permuted to fourth value and so on. The matrix after intra-row permutation is shown in Figure 5.5.

0	3	4	6	2	7	1	9	8	5
10	17	16	14	18	13	19	11	12	15
20	21	28	22	24	29	27	23	26	25
30	33	34	36	32	37	31	39	38	35
40	41	-1	42	-1	-1	-1	43	-1	-1

Figure 5.5 Matrix after Intra-row Permutation

In the last step, inter-row permutation is performed on rectangular matrix using the permutation pattern defined in the standard. Inter-row permutation pattern for this example is:

T = 4, 3, 2, 1, 0 where number of values in *T* is *R* = 5

The matrix after inter-row permutation is shown in Figure 5.6.

40	41	-1	42	-1	-1	-1	43	-1	-1
30	33	34	36	32	37	31	39	38	35
20	21	28	22	24	29	27	23	26	25
10	17	16	14	18	13	19	11	12	15
0	3	4	6	2	7	1	9	8	5

Figure 5.6 Matrix after Inter-row Permutation

Afterwards, the values in the matrix are read out column by column after pruning dummy bits to construct interleaved order of data values. An interleaved order for *L = 44* is:
Interleaved order = Π = {40 30 20 10 0 41 33 21 17 3 34 28 16 4 42 36 22 14 6 32 24 18 2 37 29 13 7 31 27 19 1 43 39 23 11 9 38 26 12 8 35 25 15 5}

b. LTE Interleaver

Quadratic Permutation Polynomial (QPP) interleaver used in LTE [LTE08] is mostly conflict free. However, for higher data rate applications when trellis and recursive units parallelism are also included in each SISO, QPP interleaver is not contention-free and requires a router and buffer mechanism to solve memory conflicts. For block size *L*, QPP interleaver is represented by following equation.

$$\Pi(x) = (f_1 x + f_2 x^2) \bmod L \qquad \ldots (1)$$

Where f_1 is odd and relatively prime to L, f_2 is a multiple of an arbitrary selected prime factor of L, x and $\Pi(x)$ represents the original and interleaved address respectively and integers f_1, f_2 are different for different block lengths defined in the standard as shown in Table 5.2.

Table 5. 2. Turbo code interleaver parameters

i	*L*	f_1	f_2	*i*	*L*	f_1	f_2	*i*	*L*	f_1	f_2	*i*	*L*	f_1	f_2
1	40	3	10	48	416	25	52	95	1120	67	140	142	3200	111	240
2	48	7	12	49	424	51	106	96	1152	35	72	143	3264	443	204
3	56	19	42	50	432	47	72	97	1184	19	74	144	3328	51	104
4	64	7	16	51	440	91	110	98	1216	39	76	145	3392	51	212
5	72	7	18	52	448	29	168	99	1248	19	78	146	3456	451	192
6	80	11	20	53	456	29	114	100	1280	199	240	147	3520	257	220
7	88	5	22	54	464	247	58	101	1312	21	82	148	3584	57	336
8	96	11	24	55	472	29	118	102	1344	211	252	149	3648	313	228
9	104	7	26	56	480	89	180	103	1376	21	86	150	3712	271	232
10	112	41	84	57	488	91	122	104	1408	43	88	151	3776	179	236
11	120	103	90	58	496	157	62	105	1440	149	60	152	3840	331	120
12	128	15	32	59	504	55	84	106	1472	45	92	153	3904	363	244
13	136	9	34	60	512	31	64	107	1504	49	846	154	3968	375	248
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146
25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1056	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1088	171	204	141	3136	13	28	188	6144	263	480

However, the equation (1) includes multiplication and square function to be implemented which is hard to embed online. So, the on-line calculation approach [SUN11] rewrites (1) into the following recursive form:

$$\Pi(x+1) = f_1(x+1) + f_2(x+1)^2 \bmod L \quad \ldots (2)$$

$$= \Pi(x) + g(x) \quad \bmod L \quad \ldots (3)$$

where $g(x) = f_1 + f_2 + 2f_2x$ which can also be computed recursively through:

$$g(x+1) = g(x) + 2f_2 \bmod L \quad \ldots (4)$$

Example for LTE Interleaver

Let us consider an example for $L = 4$. It can be seen from $f_1 = 7,\ f_2 = 12$ *So,*

$f(x) = 0,\ g(0) = 7+12 = 19,$

$f(1) = 0 + 19 \bmod 48 = 19$ *and* $g(1) = 19 + 24 \bmod 48 = 43$

$f(2) = 19 + 43 \bmod 48 = 14$ *and* $g(2) = 43 + 24 \bmod 48 = 19$

So the complete set of interleaved order can be generated recursively in the above mention procedure. The complete interleaved order is given below:

Π = {0 19 14 33 28 47 42 13 8 27 22 41 36 7 2 21 16 35 30 1 44 15 10 29 24 43 38 9 4 23 18 37 32 3 46 17 12 31 26 45 40 11 6 25 20 39 34 5}

The second step generates the conflict free memory mapping by executing memory mapping approach which is explained below.

2.3. Execution of memory mapping approach

The second step of the design flow is the execution of the memory mapping approach. The proposed work in chapter-4 is not considered here as that is our recent work (completed in last year of my thesis work) which is not published yet. So, the proposed approaches of chapter-4 are included in the future perspectives of this work. We have embedded on-chip the polynomial time algorithm presented in [SAN13]. This approach is described briefly.

	t_1	t_2	t_3	t_4
PE_1	0	1	2	3
PE_2	4	5	6	7
PE_3	8	9	10	11

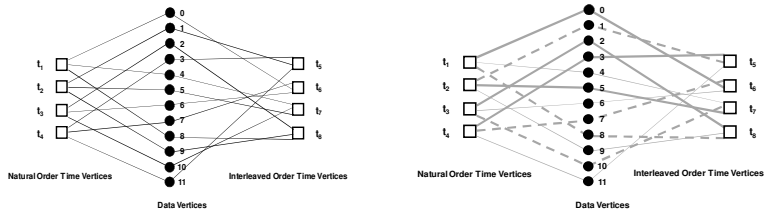
Natural Order

	t_5	t_6	t_7	t_8
PE_1	3	7	4	2
PE_2	1	0	10	9
PE_3	11	6	5	8

Interleaved Order

Figure 5.7 Data access matrix

The algorithm is based on two steps. In the first step a bipartite graph is constructed based on two data access matrices. Whereas in the second step a polynomial time bipartite edge coloring algorithm is used to find conflict free memory mapping. In the first step in order to construct a bipartite graph, a tripartite graph $G' = (T_{NAT} \cup T_{INT} \cup L, E)$ is constructed based on natural and interleaved data access matrices (as example shown in Figure 5.7) in which vertex sets T_{NAT} and T_{INT} represent all the time instances used in natural order access and interleaved order access respectively whereas vertex set L represents all the data elements used in the computation. An edge (t_{NAT}, l) is incident to the data vertex l and to the natural order time vertex t_{NAT} if l needs to be processed at t_{NAT} (i.e. data l will be read and next written at time t_{NAT}). Similarly, an edge (t_{INT}, l) is incident to the data vertex l and to the interleaved order time vertex t_{INT} if l needs to be processed at t_{INT}. This tripartite graph Figure 5.8(a) is converted into bipartite graph G by first joining two edges at each data vertex and then removing all the data vertices from the tripartite graph. G is regular with the degree of each time node, $k=P$.



(a) Tripartite graph (b) Complete Edge Coloring

Figure 5.8 Bipartite Edge Coloring Algorithm

After constructing bipartite graph, the next step is to apply bipartite edge coloring algorithm to color the edges of that graph in polynomial time. For this purpose, we find an Euler partitioning by taking every other edge to obtain two $(k/2)$-regular subgraphs. In this way the problem is reduced to two $(k/2)$-regular graphs. However, to find euler partitioning, it is necessary that k is even to divide a regular graph into two regular subgaphs of equal degree. So, if K is odd then the algorithm first finds perfect matching Mp in G, assign one color to the edges of Mp and remove Mp from G. The problem is reduced to even $(K-1)$-regular graph. The perfect matching algorithm runs in $O(kD)$ time. The complete edge coloring of G' after attaching data vertices in G is shown in Figure 5.8(b). In this figure, three colors of the edges, corresponds to three memory banks, are represented with gray bold, gray narrow and gray dotted lines. Further details on this algorithm can be found in [SAN13]. The resultant memory mapping is:

Bank $b_0=\{0,2,3,5\}$, *Bank* $b_1=\{1,7,8,10\}$, *Bank* $b_2=\{4,6,9,11\}$

Afterwards, addressing and network control logic are generated based on this mapping and stored in the memory. So, if we change the interleaving law then we get a new mapping that is different from the previous one using memory mapping approach. For example, new interleaved order and memory mapping are:

Interleaved order = 2, 7, 10, 8, 9, 6, 1, 5, 11, 3, 4, 0.

Bank b_0={0, 1, 2, 3} Bank b_1={4, 5, 6, 11}, Bank b_2={7,8 9,10}

2.4. Routing Algorithm

The third step of the design flow of our proposed approach is the execution of the routing algorithm to generate the routing information for the targeted interconnection network. We have considered two fully connected networks: Crossbar and Benes network.

Crossbar is a non-blocking network in which connection of a processing element to a memory bank does not interfere the connection of any other processor to any other memory bank. It is feasible for online approaches to use crossbar network due to its high speed as they can be configured automatically. But, their use is normally limited to low level of parallelism, due to high complexity and cost. Hardware constraints such as the number of available pins and the available wiring area limits the number of physical connections of a switch. These issues prevent the use of crossbar networks for large network sizes [DUA03]. For a Crossbar network, the size of network_RAM= T^* (B *$\log_2 B$).

Benes network needs a routing algorithm to generate routing information. For this purpose, we define a simplified routing algorithm for Benes network that can be executed on-chip along with the mapping algorithm.

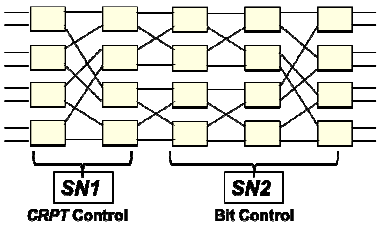


Figure 5.9 SN1 and SN2 for the routing algorithm

In a Benes network, many useful permutations, often required in parallel processing environments are found to be self-routable. Lenfant proposed efficient set-up algorithms for some frequently used permuatation as bijections [LEN78], namely the FUB family. Nassimi and Sahni [NAS81] proposed a simpler algorithm for routing the F class of permutations that includes the bit-permute complement (BPC) and inverse omega classes of permutations.

Boppana and Raghavendra [BOP88] developed another self-routing technique for linear-complement (LC) class and inverse omega class of permutations. A control algorithm is described in [NAS82] for the Benes network. This control algorithm, called the "looping algorithm," is based upon the recursive configuration of the Benes network. We have adopted to implement the control algorithm presented in [LEE87] which is not recursive. In this algorithm (B x B) Benes network is viewed as a concatenation of two sub-networks *SN1* and *SN2* as shown in Figure 5.9. The first (log B - 1) stages of a Benes network correspond to *SN1*, and the remaining log B stages correspond to *SN2*. The control algorithm sets switches one stage at a time, stage by stage. *SN1* is controlled by a full binary tree of set partitioning functions, called a Complete Residue Partition Tree (*CRPT*) and *SN2* is bit controlled. In *CRPT* of SN1, a *Complete Residue System* modulo m *CRS*-(mod m) is a set of m integers which contains exactly one representative of each residue class mod m. Whereas, a *Complete Residue Partition* (*CRP)* is a partition is a partition in which each *CRS* (mod 2^k) is further divided into two *CRS*'s(mod $2^{(k-1)}$), k>0.. *CRP* is done at each stage in which each *CRS* -(mod m) is divided into further sets of *CRS*-(mod *m-1*) as shown in Figure 5.10.

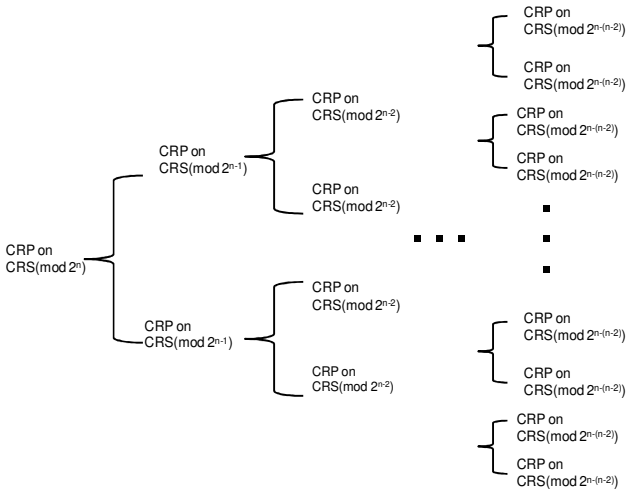


Figure 5.10 A Complete Residue Partition Tree (*CRPT*)

The Control Algorithm for the (B x B) Benes Network, $B = 2^n$ is described as follow:

1. The B numbers of the destination permutation in the binary representation are the input to the network.

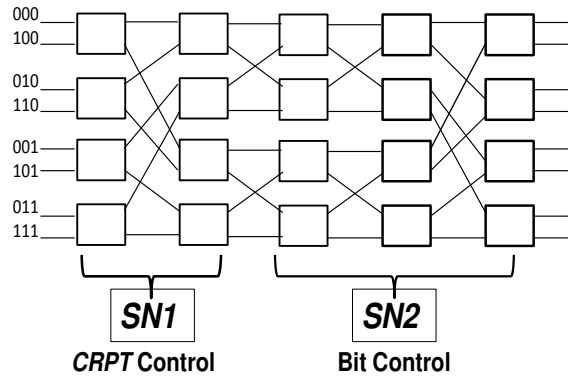
2. For *SN1*, perform 2^i *CRP*'s on the 2^i *CRS*'s(mod $2^{(n-i)}$), formed by bits $b_i \ldots b_{(n-1)}$ for E^i stage of SNI to get $2^{(i+1)}$ *CRS*'s(mod $2^{(n-i-1)}$) using bits $b_{(i+1)} \ldots b_{(n-1)}$ for $0 <= i <= (n-2)$.

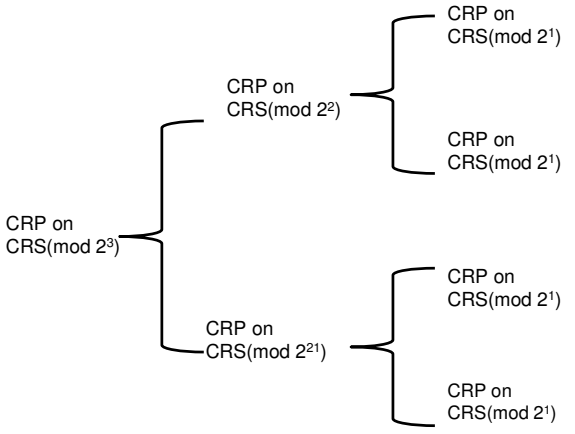
3. For *SN2*, the remaining n switching stages are controlled by $b_{(n-1)}, b_{(n-2)}, \ldots, b_0$ used as control bits $C_o, C_1, \ldots, C_{(n-1)}$, respectively.

2.4.1. Example for Routing Algorithm

Let us consider an 8x8 Benes network with a permutation [0 4 2 6 1 5 3 7] to be routed using the above defined algorithm.

The network is divided in *SN1* and *SN2* as shown in Figure 5.11(a). *SN1* consist of first two stages and *SN2* consist of last three stages. For *SN1,* the division of stages in the *CRPT* is shown in Figure 5.11(b) in which the permutation is shown in binary form controlled by three digits b_2, b_1 and b_0. In the first stage E^O of *SN1*, *CRP* is applied on *CRS* (mod 8) which is divided in two *CRS*(mod 4) using two bits bits b_1b_2 i-e (000, 010, 101, 111) and (100, 110, 001, 011) see Figure 5.10. In the second stage E^1 of *SN1*, *CRP* is applied on each *CRS* (mod 4) which divide each of it independently in two further two sets of *CRS*(mod 2) using one bit b_2 i-e (000, 101), (010, 111), (100, 001), and (110, 011).





(a) (b)

Figure 5.11 Routing example

The complete control mechanism is shown in Figure 5.12. The control mechanism for the last three stages of *SN2* are based on bit control as shown in the figure. In *SN2,* the first stage is controlled by C_0, the second stage is controlled by C_1 and the third stage is *SN2* is controlled by C_2. In *SN,* the switch is set straight if the inputs to the switch are (0,0) or (0,1) and the switch is set cross if the inputs to the switch are (1,0) or (1,1).

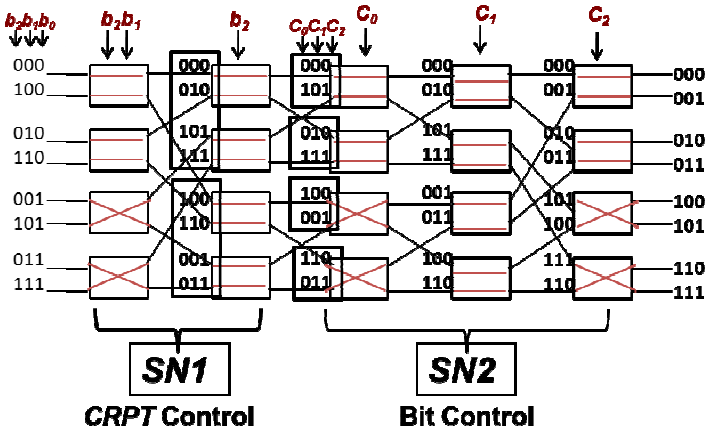


Figure 5.12 Example with complete routing tags

3. Experiments

In this section, different experiments are presented. These experiments are performed by using different embedded processors to measure the computational complexity of memory mapping approaches based on block size and parallelism. Moreover, the memory required to store command words both in case of on chip and off chip execution of memory mapping approaches is also compared. For experimental purpose, one hard processor PowerPC embedded in Xilinx FPGA and one soft processor NIOS-II used in Altera FPGAs are considered to execute the approach proposed in [SAN13] and the results are compared with other state of the art approaches [TAR04] [CHA10a]. The execution times for these approaches are measured for different processors. Moreover, HSPA interleaver used in 3GPP-WCDMA [HSP04] is implemented on parallel architecture. In this thesis we have mostly chosen HSPA interleaver for our experiments because the number of conflicting data elements in this interleaver is high as compared to other interleavers. To design parallel conflict free architecture for this interleaver, memory mapping approaches are required to generate commands word for network and memory to support all the block sizes with different parallelisms.

The first processor we considered for the experiments is PowerPC which is a hard processor embedded in Xilinx Virtex-5 ML507 board. Processor clock frequency of 400MHz and System clock frequency of 100MHz was used to perform experiments. The second processor we considered in our experiments is NIOS II. NIOS II is a soft processor used in Altera FPGAs. NIOS II has been implemented on Cyclone-III NIOS II Embedded Evolution Kit with Processor clock frequency of 195MHz and System clock frequency of 50MHz. Normalized time values (as each embedded processor has different frequency so we have

normalized all the values to PowerPc) are used to measure the impact of architecture of embedded processors on execution time. PowerPC execution time is used as a reference for normalized time and execution times of NIOS II is normalized with respect to the PowerPC clock frequencies.

The normalized times to execute (only the time needed to execute the memory mapping approach) [TAR04] [CHA10a] and [SAN13] on embedded processors for different L with P =4,8,16 and 32 are studied. The normalized times to execute [TAR04] [CHA10a] and [SAN13] for different L with P =32 are shown in Figure 5.13 for NIOS II and PowerPC. From processor perspective, PowerPC executes the mapping algorithm in the least time as compared to NIOS II. From this figure, it is evident that significant reduction in execution time for [SAN13] is achieved as compared to [TAR04] [CHA10a] for all block lengths. For L = 5120 with P = 32, using PowerPC the execution time in case of [TAR04] is same as [CHA10a] which is about 2 hours whereas by using the approach [SAN13] the execution time is reduced significantly to only 127ms. Furthermore, the results for [SAN13] also include the delay introduced by routing information generation process whereas the routing information is automatically generated as crossbar is considered for [TAR04] and [CHA10a]. For experimental purpose, we have considered crossbar for [TAR04] and [CHA10a] for which the time to generate routing information is very simple and automatic so we considered it negligible. An additional delay of 0.2ms (not included in the results shown in Figure 5.13 and 5.14) is required in order to generate routing information in case of [SAN13] for complete block length of size L=256 with P=*32*, which increases to 4ms for L=5120 with same P. However, for the same block length the delay in command word generation remains almost the same for different level of parallelism as explained in previous section.

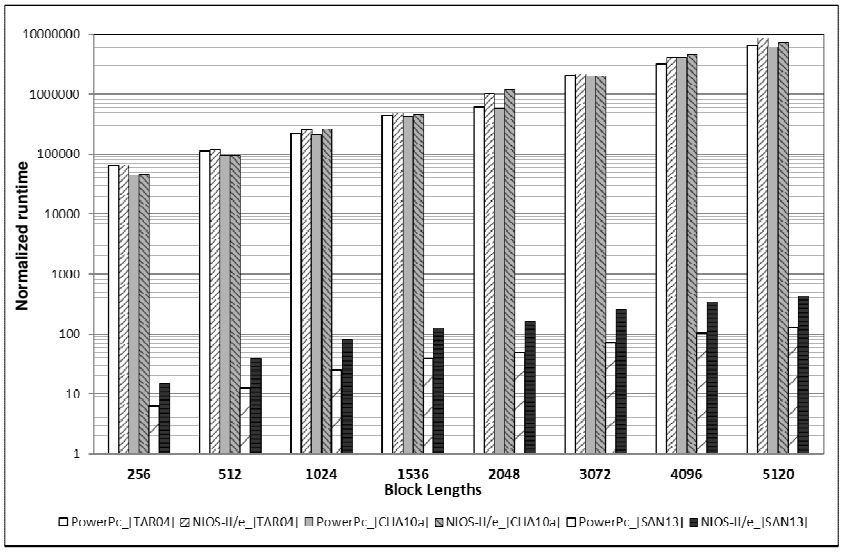


Figure 5.13 Normalized Run time Values for different embedded processors with PE = 32

Furthermore, we performed experiments with P=4, 8, 16 and 32 for L=5120 as shown in Figure 5.14. From these experiments, significant reduction in execution time for [SAN13] can be seen as compared to [TAR04] and [CHA10a]. Furthermore, the execution time for [SAN13] has no significant change with increase in parallelism whereas execution time for [TAR04] and [CHA10a] increases almost 25 times with the increase in parallelism from P =4 to P =32.

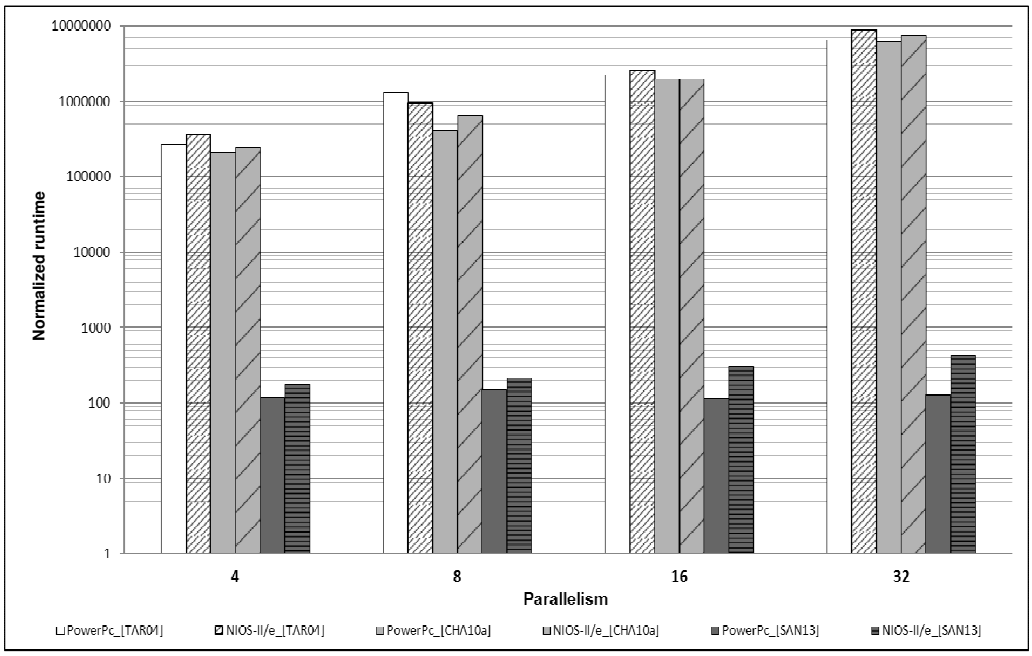


Figure 5.14 Normalized Run time Values of for different types of parallelism with *L*=5120

From architecture perspectives, cost of our architecture always remains constant for different parallelisms supporting several block lengths for each processor. However, for offline memory mapping approaches, high memory cost is required to support several block lengths. Figure 5.15 shows the comparison between the memory required to store command words with *P*=4, 8, 16 and 32. Same memory is reused to store command words as soon as the parallelism is changed in order to support this parallelism in case of proposed approach whereas off-chip approaches require additional memory to store new set of command words with each type of parallelism. For *P*=32, size of memory required in case of off-chip approach to store command words is 64-Mbits to implement all the block sizes used in 3GPP-WCDMA. Thanks to the extensive reuse of RAM only 128Kbits of memory is required in case of on-chip execution of mapping algorithms. However, we need an embedded processor for the proposed approach.

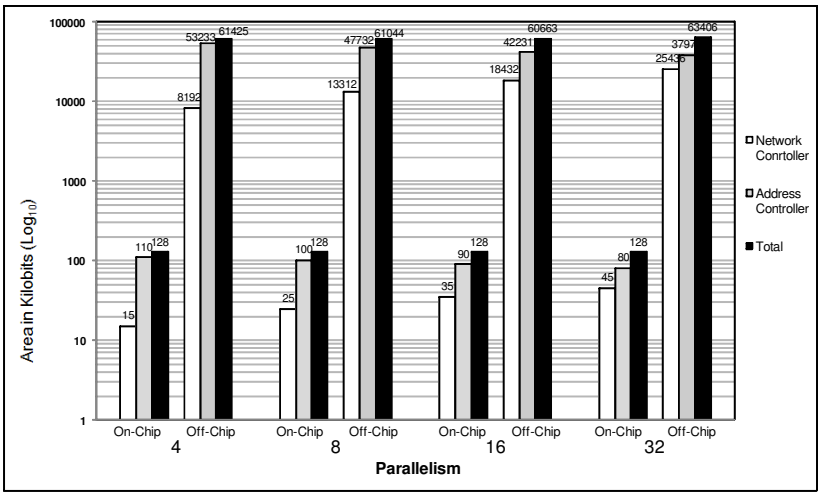


Figure 5.15 Area Comparison of on-chip and off-chip implementation for different *P*

The use of RAM blocks instead of multiple ROM blocks for storing configuration bits needed to support multiple block lengths and significant reduction in execution time in case of polynomial time approaches encourages embedding memory mapping and routing algorithm for future telecommunication devices.

4. Conclusion

In this chapter, we have proposed to embed memory mapping approaches on-chip to solve memory conflict problem in parallel hardware decoders. Dedicated architecture composed of an embedded processor and RAM blocks to store command words is proposed. We have embedded on-chip a polynomial time memory mapping approach and routing algorithm based on Benes network to solve memory conflict problem in parallel hardware decoders. Different experiments have been performed by using existing memory mapping approaches executed on several embedded processors. Results showed that the on-chip implementation of polynomial time memory mapping approaches allows to reduce significantly the execution time and reduction in hardware cost by the use of RAM blocks instead of multiple ROM blocks (though we need an embedded processor for proposed approach).

Future perspective of this work is to further improve the execution time by using ASIPs or ASICs to target real time flexible decoder architectures and to include cache like mechanism to speed up the switch from one configuration to another one. Another future perspective is to embed the polynomial time approaches proposed in this thesis (in-place memory mapping architecture approaches) to support flexible hardware architecture to support multiple block lengths and/or multiple applications.

CONCLUSION AND FUTURE PERSPECTIVES

Turbo and LDPC codes are two families of codes that are extensively used in current communication standards due to their excellent error correction capabilities. For high throughput performance, decoders are implemented on parallel architectures in which more than one processing elements decode the received data. However, parallel architecture suffers from *memory conflict problem.* It increases latency of memory accesses due to the presence of conflict management mechanisms in communication network and unfortunately decreases system throughput while augmenting system cost.

To tackle memory conflict problem, three different types of approaches exist in literature. In first type of approaches, codes are constructed with good error correction capabilities without any conflicts. Also, these interleavers often simplify the parallel decoder architectures. However, these are conflict free only for particular types or degrees of parallelism used in turbo decoding or for a subset of block lengths. A second class of solution to deal with memory access conflict problem is to simply assign data in different memory locations without considering concurrent access issue and then use additional buffers in the interconnection network to manage memory conflicts. These kinds of approaches can greatly increase the cost of the system due to presence of interconnection network and buffer management mechanism to manage conflicts. The total latency of the system is also impacted since each conflicting data access must travel buffers before being stored in the memory banks which in turn decreases the throughput. Third type of approaches are design time memory mapping approaches. The resultant architectures consist of ROM blocks used to store configuration bits. The use of ROM blocks may be sufficient to design parallel architecture that supports single codeword or single application. However, to design hardware architecture that supports complete standard or different applications, ROM based approach results in huge hardware cost and area.

In this thesis, we aimed to design optimized parallel interleaver architecture. For this purpose, we have proposed two different categories of approaches. In first category, we have proposed optimized design time off-chip approaches that aim to limit the cost of final decoder architecture targeting the customization of the network and the use of in-place memory architecture.

In the second category, we have introduced a new method in which both runtime and design time approaches can be merged to design flexible decoder. For this purpose, we have embedded memory mapping algorithms on-chip in order to execute them at runtime to solve conflict problem. The on-chip implementation replaces the ROM blocks with a single RAM block to support multiple block lengths and/or to support multiple applications. Different experiments are performed by executing memory mapping approaches on several embedded processors.

Future Perspectives

A lot of scope for future enhancement of proposed contributions is possible. In future, we can further optimize the decoder architecture by merging the two design time approaches proposed in this thesis: the network customization and in-place architecture. We can take advantage of using the customized network approach along with in-place memory mapping architecture in order to generate strongly optimized architectures. Another future perspective is to embed proposed polynomial time approaches (in-place memory mapping architecture approaches) to support flexible hardware architecture for different block lengths with in a standard or to support different standards.

In the on-chip approach, memory mapping approaches are also executed on several embedded processors. The results showed that the proposed approach allows to greatly improve timing performances and to reduce memory footprint. Future perspective of this work is to further improve the execution time by using ASIP or ASICs to target real time flexible decoder architectures and to include cache like mechanism to speed up the switch from one configuration to another one.

Furthermore, the memory mapping approaches can be used to solve memory mapping like problems from other signal processing and telecommunication domains. Implementation of different algorithms on parallel processing becomes an active domain of research after the development of high data rate applications. In all of these implementations, multiple accesses cause memory conflict problem. In future, current algorithms could be used to solve the mapping problem of other applications used in signal processing domain.

BIBLIOGRAPHY

[AHA] "Primer: Reed-Solomon Error Correction Codes", AHA Application Note.

[ALK11] R. Al-Khayat, P. Murugappa, A. Baghdadi, and M. Jezequel, "Area and throughput optimized asip for multi-standard turbo decoding," in Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on, may 2011, pp. 79 –84.

[ASG10] Rizwan Asghar and Dake Liu "Towards Radix-4, Parallel Interleaver Design to Support High-Throughput Turbo Decoding for Re-Configurability" 33rd IEEE SARNOFF Symposium 2010, Princeton, NJ, USA.

[BAH74] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inform. Theory, vol. IT-20, no. 2, pp. 284–287, March 1974.

[BEN96] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollarab, "A soft-input soft-output maximum a posteriori (map) module to decode parallel and serial concatenated codes," TDA Progress Report, vol. 42-127, November 1996.

[BEN04] A.Tarable, S.Benedetto, and G.Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", *IEEE Trans. Inf. Theory, vol.50, no.9*, Sept. 2004.

[BER93] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on, may 1993, vol. 2, pp. 1064 –1070 vol.2.

[BNS65] V.E. Benes, "Mathematical Theory of connecting network and telephone trafic", New York, N.Y.: *Academic*, 1965.

[BOH07] L. Boher, J. B. Dore, M. Helard, and C. Gallard, "Interleaver for high parallelizable turbo decoder" Proceedings of MC-SS'07, may 2007.

[BOP88] R. Boppana, C.S. Raghavendra, On self-routing in Benes and shuffle exchange networks, in: Proceedings of the International Conference on Parallel Processing, 1988, pp.196-200.

[BRI12] A. Briki, C. Chavet, P. Coussy and E. Martin, "A Design Approach Dedicated to Network-Based and Conflict-Free Parallel Interleavers", In *Proceedings of the 22th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2012*, Salt lake

City, USA, may 2012

[BRI13a] A. Briki, C. Chavet and P. Coussy, "A Memory Mapping Approach for Network and Controller Optimization in Parallel Interleaver Architectures", In Proceedings of the 23th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2013, page XX-YY, Paris, France, may 2013

[BRI13b] A. Briki, C. Chavet and P. Coussy, "A Conflict-Free Memory Mapping Approach To Design Parallel Hardware Interleaver Architectures With Optimized Network And Controller", In Proceedings of IEEE Workshop on Signal Processing Systems (SiPS), page XX-YY, Taipei : Taiwan, Province De Chine, oct. 2013

[BRU46] De Bruijn, N.G., “A Combinatorial Problem” Koninklijke Nederlandse Akademie v. Wetenschappen 49,758–764, 1946.

[CHA10a] C. Chavet, P. Coussy, P. Urard, and E. Martin, “Static address generation easing: a design methodology for parallel interleaver architectures,” in Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on, march 2010, pp. 1594 –1597.

[CHA10b] C. Chavet and P. Coussy, "A memory Mapping Approach for Parallel Interleaver design with multiples read and write accesses", *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2010.

[CHE02] J. Chen and M. Fossorier. “Density evolution of two improved bp-based algorithms for LDPC decoding”. IEEE Communication letters, March 2002.

[DAV] http://www.ict-davinci-codes.eu/

[DIN05] L. Dinoi, S. Benedetto, “Variable-size interleaver design for parallel turbo decoder architecture”, *IEEE Trans. Communication,Vol.53, No11*, 2005.

[DOB03] R. Dobkin, M. Peleg, and R. Ginosar, “Parallel VLSI architectures and parallel interleaving design for low-latency MAP turbo decoders”, Tech.Rep.2003 CCIT-TR436.

[DUA03] Jose Duato, Sudhakar Yalamanchili and Lionel Ni. “Interconnection Networks an Engineering Approach”, Morgan Kaufman Publishers, 2003,p.20-30.

[DVB08] *DVB DocumentA122.* “Frame structure channel coding and modulation for the second generation digital terrestrial television broadcasting system (DVB-T2),” 2008.

[EMM03] Emmanuel Boutillon, D. Gnaedig, V. Gaudet, P. G. Gulak, and M. Jezequel,

"On multiple slice turbo codes," 3rd International Symposium On Turbo Codes and Related Topics, pp. 343 – 346, 2003.

[FOS99] M.P.C Fossorier, M. Mihaljevic, and H. Imai. "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation". IEEE Transactions on communications, 47:673–680, May 1999.

[GIU02] A.Giulietti, L.Van Der Perre and M.Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements", *Electronics Leters, vol. 38, no. 5, pp.232–234*, Feb. 2002

[GOL61] M. J. E. Golay, "Complementary series", IRE Trans. on Info. Theory, vol. IT-7, pp. 82-87, April 1961.

[GUT10] I. Gutierrez, A. Mourad, J. Bas, S. Pfletschinger, G. Bacci, A. Bourdoux, H. Gierszal, "DAVINCI Non-Binary LDPC codes: Performance and Complexity Assessment", *proc of Future Network & Mobile Summit, Italy*, June 2010.

[HAM50] R.R. W. Hamming, "The Bell System Technical Journal," Bell Syst. Tech. J., vol. XXVI, no. 2, pp. 147–160, Apr. 1950.

[HSPA04] 3GPP, "Technical specification group radio access network; multiplexing and channel coding (FDD)" *(25.212 V5.9.0).*June 2004.

[JIN] Jing-ling, "Parallel Interleavers Through Optimized Memory Address Remapping" *IEEE Trans. VLSI Systems* vol. 18, no.6, pp.978-987, June. 2010.

[JOH10] Sarah J. Johnson, "Iterative Error Correction Turbo, Low-Density Parity-Check and Repeat–Accumulate Codes" Cambridge University Press 2010

[KIE03] Kienle, F., Thul, M. J., and When, N., 2003. "Implementation Issues of Scalable LDPC-Decoders". in Proceeding of 3rd International Symposium on Turbo Codes and Related Topics, Brest, France, 291-294.

[KWA02] J. Kwak and K. Lee, "Design of dividable interleaver for parallel decoding in turbo codes," Electron. Lett., vol. 38, no. 22, pp. 1362–1364, 2002.

[LEE87] K. Y. Lee, "A new Benes network control algorithm," IEEE Trans. Comput., vol. C-36, no. 6, pp. 768–772, June 1987.

[LEN78] J. Lenfant, Parallel permutations of data: A Benes network control algorithm for frequently used permutations, IEEE Trans. Comput. (1978) 637-647.

[LIN04] Shu Lin and Daniel J. Costello, Jr., "Error control Coding" Pearson Education, Inc 2004

[LTE08] "Technical Specification Group Radio Access Network; Evolved Universal

Terrestrial Radio Access; Multiplexing and Channel Coding (Release 8)", 3GPP Std. TS 36.212, Dec. 2008.

[MAC96] J.C. MacKay David and R.M. Neal, "Near Shannon limit performance of low density parity check codes", *Electronics letters*, July 1996.

[MAN03] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," IEEE Trans. VLSI Syst., vol. 11, no. 6, pp. 976–996, Dec 2003.

[MOU07] H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel, "Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding," in Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07, april 2007, pp. 1–6.

[MOU08] H. Moussa, A. Baghdadi, M. Jezequel, "Binary de Bruijn on-chip network for a flexible multiprocessor LDPC decoder", *45th DAC Design Automation Conference*, 2008.

[MUL06a] O. Muller, A. Baghdadi, and M. Jezequel, "Spc05-3: On the parallelism of convolutional turbo decoding and interleaving interference," in Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE, 27 2006-dec. 1 2006, pp. 1 –5.

[MUL06b] O. Muller, A. Baghdadi, M. Jezequel, "ASIP-based multiprocessor SoC design for simple and double binary turbo decoding", *DATE,* 2006.

[NAS81] D. Nassimi, S. Sahni, A self-routing Benes network and parallel permutation algorithms, IEEE Trans.Comput. 1981, 332 - 340.

[NAS82] D. Nassimi, S. Sahni, Parallel algorithm to set up the Benes permutation network, IEEE Trans. Comput. (1982) 148-154.

[NEE05] C. Neeb, M. Thul, and N. Wehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in Proc. IEEE International Symposium on Circuits and Systems (ISCAS), May 2005, pp. 1766 – 1769 Vol. 2.

[PEA88] J.Pearl, "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible reference", Morgan Kaufmann, 1988.

[QUA06] F.Quaglio, F.Vacca, C.Castellano, A.Tarable, M.G.Asera. "Interconnection Framework for High-Throughput, Flexible LDPC Decoders". In proceeding Design Automation and Test in Europe Conference and Exhibition, 2006.

[REH14a] S. Ur Rehman, C. Chavet and P. Coussy, "A Memory Mapping Approach

based on Network Customization to Design Conflict-Free Parallel Hardware Architectures", In Proceedings of the 24th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2014, page XX-YY, Houston, Texas, USA, may 2014.

[REH14b] S. Ur Rehman, A. Sani, P. Coussy and C. Chavet, "Embedding Polynomial Time Memory Mapping and Routing Algorithms on-chip to Design Configurable Decoder Architecture", In Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing, Florence, Italy, May, 2014.

[REH13] Saeed. Rehman, A. Sani, P. Coussy, C. Chavet, "On-Chip Implementation Of Memory Mapping Algorithm To Support Flexible Decoder Architecture", In Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, May, 2013.

[REH12] SANCHEZ GONZALEZ Oscar David, REHMAN Saeed Ur, SANI Awais, JEZEQUEL Michel, CHAVET Cyrille, COUSSY Philippe, JEGO Christophe "A dedicated approach to explore design space for hardware architecture of turbo decoders" SiPS 2012: IEEE Workshop on Signal Processing Systems, 17-19 october 2012, Quebec, Canada, 2012, pp. 288-293.

[ROS04] A.La Rosa, C.Passerone, F. Gregoretti, L. avagno, "Implementation of a UMTS turbodecoder on dynamically reconfigurable platform", DATE, 2004.

[SAN11a] A. Sani, P. Coussy, C. Chavet, and E. Martin, "A methodology based on transportation problem modeling for designing parallel interleaver architectures," in Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, may 2011, pp. 1613 –1616.

[SAN11b] A. Sani, P. Coussy, C. Chavet, and E. Martin, "An approach based on edge coloring of tripartite graph for designing parallel ldpc interleaver architecture," in Circuits and Systems (ISCAS), 2011 IEEE International Symposium on, may 2011, pp. 1720–1723.

[SAN12] A.H. Sani "Bipartite edge coloring approach for designing parallel hardware interleaver architecture", thesis dissertation 2012.

[SAN13] A.H. Sani, C. Chavet and P. Coussy, "A First Step Toward On-Chip Memory Mapping for Parallel Turbo and LDPC Decoders: A Polynomial Time Mapping Algorithm*", IEEE Transactions on Signal Processing, vol. 61, issue: 16, p.4127 - 4140,* 2013.

[SAC12] O. Sanchez, S. ur Rehman, A. Sani, C. Jego, C. Chavet, P. Coussy, and M. Jezequel, "A dedicated approach to explore design space for hardware architecture of turbo decoders", In Proceedings of the IEEE Workshop on Signal Processing Systems, page XXX, Quebec, Canada, October 17-19, 2012

[SHA49] Shannon, Claude E. (1949), "A theorem on coloring the lines of a network", J. Math. Physics 28: 148–151

[SOI08] Soifer, Alexander (2008), *The Mathematical Coloring Book*, Springer-Verlag

[STU11] Christoph Studer, Christian Benkeser, Sandro Belfanti, and Quiting Huang, "Design and implementation of a parallel turbo-decoder asic for 3gpp-lte," Journal of Solid state circuits, vol. 46, pp. 8–17, 2011.

[TAK06] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," IEEE Trans. Inf. Theory, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.

[VIZ64] V. G. Vizing;On an estimate of the chromatic class of a p-graph(In Russian), Diskret.Analiz. 3: 25-30, 1964

[WAN11] G Wang, Y Sun, JR Cavallaro, Y Guo, "High-throughput contention-free concurrent interleaver architecture for multi-standard turbo decoder" Application-Specific Systems, Architectures and Processors (ASAP), 2011.

[WEH02] M. Thul, N. Wehn, and L. Rao, "Enabling high-speed turbo decoding through concurrent interleaving," in Proc. IEEE International Symposium on Circuits and Systems (ISCAS), vol. 1, 2002, pp. 897–900.

[WEH02a] M. I. Thul, F. Gilbert. and N. Wehn. "Optimized Concurrent Interleaving for High-speed Turbo-Decoding". In Proc. 9rh IEEE International Conference on Electronics, Circuits and Systems - ICECS 2002, Dubrovnik, Croatia, Sept. 2002.

[WEH03] M. Thul, F. Gilbert, and N. Wehn, "Concurrent interleaving architectures for high-throughput channel coding," in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2003, pp. 613–616 vol.2.

[WIF08] *IEEE P802.11n/D5.02, Part 11*. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Enhancements for Higher Throughput", July 2008.

[WIM06] *IEEE P802.16e, Part 16*. "Air Interface for Fixed and Mobile Broadband Wireless Access Systems," Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands, and Corrigendum 1, Feb. 2006.

[WHE04] N. When, "SOC-Network for Interleaving in wireless Communications", *MPSOC*, 2004.

[WON10] Cheng-Chi Wong, Ming-Wei Lai, Chien-Ching Lin, Hsie-Chia Chang, and Chen-Yi Lee, "Turbo decoder using contentionfree interleaver and parallel architecture," Journal of Solid state circuits, vol. 45, pp. 422–432, 2010.

[WON11] Cheng-Chi Wong and Hsie-Chia Chang, "High-efficiency processing schedule for parallel turbo decoders using qpp interleaver," IEEE Transations on Circuits and Systems, vol. 58, pp. 1412–1420, 2011.

[WON10b] S Cheng-Chi Wong and Hsie-Chia Chang, "Reconfigurable Turbo Decoder With Parallel Architecture for 3GPP LTE System" IEEE Trans. Circuits Syst. II, Exp.Briefs, vol. 57, no. 7, pp. 566–570, Jul. 2010.

[WOO00] J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: An overview," IEEE Trans. Veh. Tech., vol. 49, no. 6, pp. 2208–2233, Nov. 2000.

[XIO05] Xiao-Yu Hu., Eleftheriou, E., Arnold, D.M.: 'Regular and irregular progressive edge-growth Tanner graphs', IEEE Trans. Inf. Theory, 2005, 51 pp. 386–398

[YAN11] YANG Sun and J. R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE/LTE-advance turbo decoder," INTEGRATION, the VLSI Journal, vol. 44, pp. 305-315, Jan. 2011.

[YOO02] S. Yoon and Y. Bar-Ness, "A parallel map algorithm for low latency turbo decoding," IEEE Commun. Lett, vol. 6, no. 7, pp. 288–290, 2002.

[ZHA04] Y. Zhang and K. Parhi, "Parallel turbo decoding," in Proceedings of the ISCAS '04., vol. 2, 23-26 May 2004, pp. 509–512 Vol.2.

[ZHA05] Juntan Zhang and Fossorier M.P.C, "Shuffled iterative decoding," IEEE Trans. on Communications, vol. 53, pp. 209–213, 2005.

[ZHA07] Juntan Zhang, Yige Wang, and Marc P. C. Fossorier, "Iterative decoding with replicas," IEEE TRANSACTIONS ON INFORMATION THEORY, vol. 53, pp. 1644–1663, 2007.

PERSONAL BIBLIOGRAPHY

International conferences

[REH14a] S. Ur Rehman, C. Chavet and P. Coussy, "A Memory Mapping Approach based on Network Customization to Design Conflict-Free Parallel Hardware Architectures", In Proceedings of the 24th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2014, page XX-YY, Houston, Texas, USA, may 2014.

[REH14b] S. Ur Rehman, A. Sani, P. Coussy and C. Chavet, "Embedding Polynomial Time Memory Mapping and Routing Algorithms on-chip to Design Configurable Decoder Architecture", In Proceedings of the 39th IEEE International Conference on Acoustics, Speech and Signal Processing, Florence, Italy, May, 2014.

[REH13] Saeed. Rehman, A. Sani, P. Coussy, C. Chavet, "On-Chip Implementation Of Memory Mapping Algorithm To Support Flexible Decoder Architecture", In Proceedings of the 38th IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, May, 2013.

[REH12] SANCHEZ GONZALEZ Oscar David, REHMAN Saeed Ur, SANI Awais, JEZEQUEL Michel, CHAVET Cyrille, COUSSY Philippe, JEGO Christophe “A dedicated approach to explore design space for hardware architecture of turbo decoders” SiPS 2012: IEEE Workshop on Signal Processing Systems, 17-19 october 2012, Quebec, Canada, 2012, pp. 288-293.

National conference

[REH14c] S. Ur Rehman, C. Chavet and P. Coussy, " Designing optimized parallel interleaver architecture through network customization", In Colloque national du GDR SoC-SiP, Paris, France, june 2014.

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
B	Number of banks
BEN	Benes network
BS	Barrel shifter network
BF	Butterfly network
BFS	Breadth first search
CB	Crossbar network
c_i	A color in Vizing theorem
CN	Check node
CRPT	Complete Residue Partition Tree
CRS	Complete Residue System
DVB-S	Digital Video Broadcasting — Satellite
FEC	Forward error correction
GIBB	General Interleaver Bottleneck Breaker
g	Size of girth in NB-LDPC
GPP	General Purpose Processor
HSPA	High Speed Packet Access
LDPC	Low Density Parity Check
LLR	Log likelihood Ratio
LTE	Long-Term Evolution
L	Block length
l	Additional data elements
M	Size of a memory bank
MRMW	Multiple read multiple write
NB-LDPC	Non-binary Low Density Parity Check
NoC	Network on chip
OPMM	optimized memory address remapping
PE	Processor element
P	Total number of processors/Parallelism
QPP	Quadratic Permutation Polynomial
R	Code rate
RIBB	Ring Interleaver Bottleneck Breaker
sf_{cur}	Current semi 2-factor
SNR	Signal to noise ratio

SoC	System on Chip
TIBB	Tree Interleaver Bottleneck Breaker
T	Total number of time instances
t_i	A time instance
UWB	Ulta wide band
VN	Vector node
WiMAX	Worldwide Interoperability for Microwave Access
WiFi	Wireless local area network products that are based on IEEE 802.11 standards
π	Interleaver law